University of Southern California

Requirements Envisaging by Utilizing Scenarios (REBUS)

Lorna A. Zorman

USC/Information Sciences Institute

August 1995

ISI/RR-95-430

INFORMATION SCIENCES INSTITUTE

310/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

Requirements Envisaging by Utilizing Scenarios (REBUS)

Lorna A. Zorman

USC/Information Sciences Institute

August 1995

ISI/RR-95-430

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** August 1995	**3. REPORT TYPE AND DATES COVERED** Research Report

4. TITLE AND SUBTITLE

Requirements Envisaging by Utilizing Scenarios (REBUS)

5. FUNDING NUMBERS

ARPA/ITO:
F33615-94-1-1402
U. of Michigan:
Subcontract PO# V08985
Rome Laboratory:
F30602-89-C-0103

6. AUTHOR(S)

Lorna A. Zorman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

ISI/RR-95-430

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

ARPA/ITO
3701 N. Fairfax Drive
Arlington, VA 22203

Avionics Directorate
Wright Laboratory
2185 Avionics Circle, Bldg. 620
Wright-Patterson AFB, OH 45433

(CONTINUED ON FOLLOWING PAGE)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Requirements envisaging is the process of transforming vague and informal requirements into precise descriptions. Envisaging evokes ideas,
project criteria, and the explanation of alternative solutions which are refined or discarded. At the envisioning stage of system development, complex
systems are typically described in a fragmentary and highly contextual manner. This conflicts with the abstract and decontextualized formal lan-
guages used by software experts. As a consequence, requirements envisaging, which seeks to bridge this gap, is a challenging phase of system devel-
opment in which to provide automated support.
In requirements envisaging, domain experts will often convey partial descriptions of system and environment behavior arising in restricted situa-
tions, namely, scenarios. Scenarios play an important role in envisaging by mediating communication and by describing alternative situations and
rationale explored during design. Despite this importance, scenarios are not, in general, formally captured as part of requirements documentation.
This dissertation is a step toward automated support for envisaging with scenarios. For this task, the representation used to capture scenarios must
support human-tool collaboration. The tool described herein supports capturing scenarios in a formal manner despite their fragmentary and contex-
tual nature. The goal is to let people who are not necessarily computer experts create scenarios easily and allow other people to readily understand
the concepts conveyed in these scenarios.
The main accomplishments reported in this thesis are: an observational study of domain and software experts utilizing scenarios, the development
of a formal representation for scenarios, an automated tool that allows people to create scenarios in that representation, and evaluation of the repre-
sentation and tool in a real world domain outside those studied during development.
These accomplishments are a step toward: bridging the communication gap between domain experts and system design experts, moving some of
the burden of work from people to machines, the documentation of domain knowledge and rationale, and the traceability between requirements and
implementation by providing a formal means to capture scenarios.

14. SUBJECT TERMS

knowledge acquisition, representations, requirements engineering, scenarios, tools, use cases

15. NUMBER OF PAGES

170

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Report Documentation Page, Form SF298, Continued

9. Sponsoring/Monitoring Agency Name(s) and Addresses, continued

University of Michigan
2044 Wolverine Tower
3003 S. State Street
Ann Arbor, Michigan 48109-1273

Rome Laboratory (C3CA)
525 Brooks Road
Griffiss AFB, NY 13441-4505

REQUIREMENTS ENVISAGING BY UTILIZING SCENARIOS (REBUS)

by

Lorna Ann Zorman

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 1995

Acknowledgements

I would like to express my thanks to the people who encouraged and aided me in this work:

My Advisor and committee: My advisor, Lewis Johnson, helped give me direction when I needed it most. He worked with me to develop an interesting and enjoyable thesis topic. Martin Feather, who unofficially joined my committee when I moved in to the office next door and officially joined my committee just before my defense. Paul Rosenbloom, for providing the opportunity to study air-combat scenarios and for his diligence toward clarity and evaluation. Michael Noll, an outstanding external counselor and teacher.

Proofreading: Brian Lau for spending his free time removing superfluous little words and phrases from this dissertation.

The folks at JPL: Trish Santos, Roland Bevan, Richard Chen, and Randy Hill.

Friends, collegues, helpers, scenario folks, etc. Richard Angros, Bob Balzer, Kevin Benner, Dave Bridgeland, Don Cohen, Ali Erdem, Yolanda Gil, Neil Goldman, Raymonde Guindon, Bob Hall, Ellis Horowitz, Ken Kahn, John Karat, Patti Koenig, Robin Lampert, Yingsha Liao, Ping Luo, Lee Magnone, Tony Marston, Chris McClenaghan, Linda Mizushima, Bonnie Nardi, Robert Neches, Colin Potts, Shankar Rajamoney, Pam Rothman, Jim Rhyne, Kenny Rubin, Rodney Ruddock, Tom Russ, Karl Schwamb, John Stasko, Sergio Socarras, K. Swamy, Bill Swartout, Pedro Szekely, Milind Tambe, Brenda Timmerman, Dave Wile, Tom Wisniewski, Cathy Wolf, Karen Zand.

The GoPATH team at Bull S.A.

The people I met at various CHI, CSCW, IWSSD, and OOPSLA conferences and workshops.

My Family: I'd like to thank the two people who helped me most – my parents, Rosalie and Maurice Zorman. They provided the freedom to let me pursue whatever directions I chose, as well as the support and encouragement when times were tough. I'm very proud to dedicate this dissertation to them and my brothers, Barry and Jack, and my sister-in-law, Caren.

Contents

List Of Figures

List Of Tables

Abstract

Requirements envisaging is the process of transforming vague and informal requirements into precise descriptions. Envisaging evokes ideas, project criteria, and the explanation of alternative solutions which are refined or discarded. At the envisioning stage of system development, complex systems are typically described in a fragmentary and highly contextual manner. This conflicts with the abstract and decontextualized formal languages used by software experts. As a consequence, requirements envisaging, which seeks to bridge this gap, is a challenging phase of system development in which to provide automated support.

In requirements envisaging, domain experts will often convey partial descriptions of system and environment behavior arising in restricted situations, namely, scenarios. Scenarios play an important role in envisaging by mediating communication and by describing alternative situations and rationale explored during design. Despite this importance, scenarios are not, in general, formally captured as part of requirements documentation.

This dissertation is a step toward automated support for envisaging with scenarios. For this task, the representation used to capture scenarios must support human-tool collaboration. The tool described herein supports capturing scenarios in a formal manner despite their fragmentary and contextual nature. The goal is to let people who are not necessarily computer experts create scenarios easily and allow other people to readily understand the concepts conveyed in these scenarios.

The main accomplishments reported in this thesis are: an observational study of domain and software experts utilizing scenarios, the development of a formal representation for scenarios, an automated tool that allows people to create scenarios

in that representation, and evaluation of the representation and tool in a real world domain outside those studied during development.

These accomplishments are a step toward: bridging the communication gap between domain experts and system design experts, moving some of the burden of work from people to machines, the documentation of domain knowledge and rationale, and the traceability between requirements and implementation by providing a formal means to capture scenarios.

0

Chapter 1

Introduction

Human intelligence almost always thrives on context while computers work on abstract numbers alone. A subtraction problem cast in terms of apples is easier for a student, but a programmer must do precisely the opposite thing, convert concrete problems into context-free mathematics, because that is easier for the machine. - *Arno Penzias 1989, p.49*

Requirements envisaging is the process of transforming peoples' informal notions of what is desired into a precise description. These descriptions may then be suitable for mediating communication with oneself over time, one's own community, other communities, and even automated tools.

During envisaging people are best able to describe complex systems in a fragmentary and highly contextual manner. People need the context. This conflicts with the abstract and decontextualized formal languages used by software experts. In particular, what people state in a fragmentary and contextual manner is not entirely ready to be transformed into a decontextualized formal language. As a consequence, requirements envisaging, which seeks to bridge this gap, is a challenging phase of system development in which to provide automated support.

During envisaging, people can and do easily express *scenarios* which are partial descriptions of system and environment behavior arising in restricted situations. That is, people are able to think about the desired behavior in terms of situations that might arise, by stating when they might arise, and by stating what ought to happen or ought not to happen in those situations. In general, scenarios are not formally captured as part of the requirements documentation although scenarios play an important role in envisaging by mediating communication and by describing alternative situations and rationale explored during design.

This dissertation is a step toward providing automated support for envisaging with scenarios. In order to do this, the representation used to capture scenarios must be understandable by automated tools. The tool should support capturing scenarios in a formal manner despite their fragmentary and contextual nature. It is also important that such a tool let people who are not necessarily computer experts be able to create scenarios easily, and that other people be able to understand the concepts conveyed in these scenarios.

The benefits that are expected to accrue from providing automated support include: better communication between people, particularly domain experts and software experts; better distribution of work between people and machines by providing automated support for manipulation and analysis; better preservation of the design history and rationale for others to understand the details considered and the context of use; better traceability between the requirements and the implementation since having a formal means to capture scenarios can provide the inputs into the next generation of automated tools for software engineering.

As a step towards such benefits, the main accomplishments reported here are

- an observational study of scenario-based communication between software experts and domain experts.

- the development of a domain-independent representation for scenarios

- an automated tool allowing creation of scenarios in that representation

- evaluation of the representation and tool in a real-world context which was not studied as part of development.

1.1 Why scenarios

Scenarios are used in such diverse fields as architecture, engineering, and human factors. In such fields, requirements envisaging involves detailed thought about something wanted or needed. Envisaging is an important part of design which evokes ideas and criteria with which alternatives are explored, refined or discarded. Researchers, trying to understand the nature of individuals engaged in design, have reported that people will naturally engage in scenario activity when detailed thought is required by their subjects [31, 37]. From separate protocols of architects, mechanical engineers, and instructional trainers engaged in design, Goel coins the phrase "scenario immersion" to describe the "frequently occurring episodes in which designers recall and immerse themselves in rich, intricate images from their past experience." Furthermore, he states that across task domains and external representations (suitable for the task) the scenario episodes seemed to play a crucial role in the generation and evaluation of the design [31].

For software design, second generation object-oriented methods [49, 32, 88, 91] and the user interface design community [13, 55, 3] advocate the use of scenarios. These communities advocate the use of scenarios in the context of a variety of project tasks (e.g. user interface design, requirements acquisition, test case generation). In addition, they advocate various informal representations (e.g. use cases, scripts, story-boards) and formal representations (e.g. interaction diagrams, path expressions, message sequence charts).

The informal representations contain much more useful information than the existing formal representations were designed to capture. That is, the formal representations have reflected the abstract decontextualized languages, rather than concrete detailed scenarios. This is especially problematic when domain experts communicate detail which should be considered but is never formally captured by software experts.

Given the uses of scenarios in diverse fields, one contribution of this work is a unifying representation of scenarios. This representation is called, REBUS, which stands for Requirements Envisaging By Utilizing Scenarios.[1] The representation is formal, yet captures useful domain knowledge conveyed by domain experts. REBUS is intended to unify the best properties of the various informal and formal representations by being a representation which meets the desiderata described in chapter 2.

1.2 Who writes scenarios

The typical practice in software engineering is for systems analysts to develop abstract requirements documentation based on discussions with domain experts. Generally, an analyst has some experience in the application domain and with software systems, but does not have all the needed expertise in either. The typical documentation tends to abstract away many concrete details needed for people to understand the domain.

For example, domain experts including air-traffic controllers were a part of the design and review team for the Federal Aviation Administration's 3.6 billion dollar Advanced Automation System [45]. The typical requirements documentation was produced, though requirements changes continued to plague the project [94]. The project is nearly a billion dollars over budget and more than two years late. Thus, in spite of serious efforts to involve domain experts, requirements envisaging proved inadequate.

In typical practice, there is little opportunity for envisaging on the part of the domain experts or software experts when reading analysts' documentation. Reading is not as active a process as writing. In this typical practice, the detailed thought doesn't occur until an implementation is developed. In the case of the AAS, requirements changes occurred as more detailed thought went into the implementation.

Consider two categories of scenario writers, domain experts and software experts. Domain experts have spent years learning both the complex vocabulary of their

[1]Webster's Dictionary defines rebus as "a representation of words or syllables by pictures of objects or by symbols whose names resemble the intended words or syllables in sound; also: a riddle made up of such pictures or symbols" (e.g. RE)

domain and how to react in the complicated situations arising in this domain. This is also true for software experts who have spent years learning the formality of programming and formal logic.

When domain and software experts communicate with each other during meetings, they predominantly use their own domain-specific vocabularies. While trying to establish connections, they will actually mix their terminology with the terminology from others' outside their area of expertise, and by doing so leave the communication open to misinterpretation.

One finds senior project members taking the lead as "translators," i.e software experts who know the domain and its vocabulary, or domain experts who have learned the software experts' vocabulary. These people are in short supply and they tend not to do much of the programming, so the knowledge the programmers need remains indirect. Domain experts and software experts need to collaborate [2] with a common representation and vocabulary when writing scenarios.

In a few application domains (for example financial analysis), natural paradigms have been devised (such as spreadsheets) that enable the domain experts to themselves play the role of software experts, i.e to specify and build many of their own, typically small, software applications. In general, however, we are far from the goal of having domain experts directly develop large, complex behavioral systems without considerable intervention and assistance from software experts. Even with spreadsheets we know that beginners tend just to use forms made by others and gradually acquire skills. They also ask software experts for help when the programming effort exceeds their desire to do it themselves [68].

Domain experts need certain skills to use any computer-based support. The assumption is that our scenario writers are familiar with using direct manipulation and forms-based computer interface techniques and that scenario writers are a subset of scenario users. These human computer interface techniques form the basis for the scenario capture tool that will be described in this dissertation.

[2] A tenet of Participatory Design.

This thesis addresses the communication problems that exist between domain experts and software experts by considering each of their needs and by considering the impact of adding computers into the communication process to assure that automated support can be utilized.

1.3 Designer activities

The dynamics of software design activity are iterative, ill-structured [93] and opportunistic [36]. The nature of any design session is that it is dynamic, and does not proceed in a strictly top-down or bottom-up fashion. In the case of systems design, the process is knowledge (or lack of knowledge) and representation (artifact) driven. The dynamic nature is manifested in the design activity of individuals or groups with single or multiple media.

Guindon reports on activities of individual system analysts engaged in the design of an elevator control system [36]. This is a domain in which the analysts were not experts, but with which they were familiar. Figure 1.1 illustrates the design activities of an individual analyst. This analyst is opportunistically shifting between the application domain (elevator control) and solution details throughout the design session. Notice that there is a continued need for domain specific considerations (the upper levels of the figure) as well as continued envisaging activity into the solution details.

Shifting focus to *groups* of designers involved in design meetings, clarification is a time consuming activity. Olson et al. state [75], "clarification of ideas – a cross-cutting classification – took one third of the time, indicating how much time was spent in both orchestrating and sharing expertise among group members." Furthermore they state, "Clarification time is interesting. On the one hand, it certainly represents a coordination activity, in that it represents time devoted to establishing common ground (e.g., Clark & Brennan, 1991). On the other hand, it also contributes to the problem solving involved in design, because it helps the participants develop their ideas and make them clearer."

Figure 3. **Shifts in design activities and levels of abstraction of Designer 1. Plus signs indicate newly inferred or added requirements. Light bulbs indicate sudden discovery of partial solutions or requirements. The region marked by *R* indicates the period of solution review.**

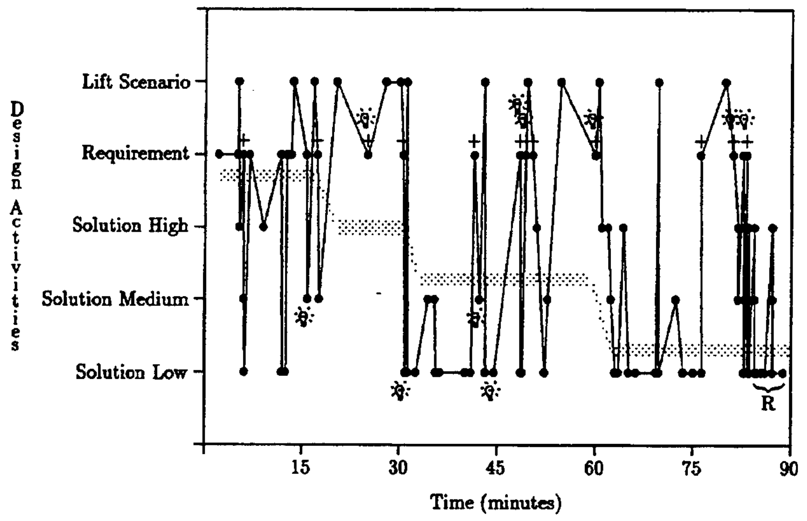


Figure 1.1: The shifting of a designers focus over time. Reprinted by permission of Lawrence Erlbaum Associates, Inc. From Guindon [36], figure 3, p. 319.

Introducing a new media or methodology for communication changes the nature of the collaboration. A general consensus is that multiple media and methodologies are needed with an understanding of when to use them [54]. Further studies by Olson et al. were performed to see how automated support affects group design meetings [77]. They studied how a synchronously shared text editor changed the character of a design meeting as well as its outcome [76]. They report that the designs produced by the groups supported by the editor were of a higher quality than those who worked with conventional white-board and paper and pencil. They were surprised by the fact that those supported by the tool did *less* extensive exploration of the design space. The tool helped the group keep more focused on the *core* issues in the emerging design, to waste less time on less important topics, and to *capture* what was said as the discussion progressed.

This thesis introduces a new media into the collaborative process of scenario writing. Although I will not be reporting on a controlled study like the Olson's, I will be describing a formative evaluation done in the context of a real world project.

1.4 Automated support

Developing automated support for scenario writing is a central contribution of this thesis. Tools specifically for scenarios don't generally exist. As a result scenarios are rarely captured in any automated tools, and even fewer are captured in a formal manner. In software practice, the few scenarios that are captured are for the purposes of testing or user documentation.

There are tradeoffs of different forms of automated tools: we can look at the variety of tools that are useful for general or domain-specific tasks as well as consider tool characteristics. A tool can be characterized along the following scales: *learnability* which depends on the scenario writers background and skills; *expressivity* from the standpoint of what concepts can be explicitly represented in a scenario; and *formality* which depends on the syntax and semantics available for scenario analysis. These characterizations are not easily quantifiable, but they serve as a means to distinguish between different forms of support and to discuss the tradeoffs. To our knowledge, previous work has not adequately resolved these tradeoffs in a manner which meets the needs of people during requirements envisaging.

What is currently used most are domain-independent, task support tools (e.g. a drawing program, which can support structured graphics for drawing aircraft in different air-spaces). Such tools are completely independent of application domain knowledge and have relatively high learnability. They vary from editing text and graphics, to outlining and story-boarding, to composing animations and multi-media presentations. Unfortunately, these tools do not support formality. Therefore, the tools leave domain specifics open to misinterpretation when such presentations are passed along to programmers. For example, a air-traffic controller might draw an aircraft in-route to LAX with a drawing tool. In such a depiction, problems arise when the attributes associated with the aircraft (such as, wide-body) can not be formally tied to the aircraft.

To pass designs along to programmers, computer-aided software engineering (CASE) tools are generally advocated. If we consider programming as a domain of expertise then CASE tools are a form of domain-specific (i.e. software design) task supporting tool with higher degrees of formality. Their learnability is limited to software experts since they require substantial programming expertise. CASE tools and object-oriented methods employ specialized vocabularies and representations. Such vocabularies are arcane to domain experts. These representations are also biased towards the expression of abstractions, inhibiting their usefulness at capturing the contextual details and semantics of the application domain knowledge.

Some tools were developed for recording design rationale (e.g. IBIS [19, 84] and QOC [61]). The goal of capturing design rationale is to retain the "why" of the design. Such tools provide hypertext support for structuring links to nodes which separate the design space into issues, alternatives, and criteria. Scenarios can be placed in the nodes, complementing the design rationale. The structured links and nodes have little value in terms of automated analysis of the design. Recent work focuses on integrating design rationale with scenarios because design rationale had limited value without being attached to the designed artifact [82].

Alternatively, approaches that take advantage of a-priori knowledge of the application domain exist. They rely upon software experts to build domain-specific tools for scenarios [25, 56, 11] (e.g. a fighter-plane simulator which has built-in plane objects and behavior for firing missiles) . While the result, if well designed, is eminently acquirable by domain experts, such a tool is of little use for a different domain, and of course requires that the software experts must already have understood the domain in order to have built the tool.

As part of the evolution of the ideas presented in this thesis, I considered visual languages and end-user programming tools, especially those which made use of demonstrational techniques for specification. Taking a domain specific approach, the initial effort was to provide support for animation with domain specific objects. In this context a prototype was developed for vehicle traffic-control which was implemented by integrating the ARIES specification environment [52] which supports various specification languages and the Polka [99] framework designed for novice programmers to develop their own algorithm animations.

This initial effort developed with the idea that it would have to be useful for critiquing behavior. The problem arose that behavior was encoded in parameterized procedures, difficult for non-programmers to understand. It became clear that this approach had two other major problems: (1) it required a great deal of domain knowledge by the software experts and it would be codified in a manner no longer suitable for envisaging; (2) a great deal of effort would be spent in coding scenarios. The effort in writing parameterized procedures for every object's behavior was especially problematic. Software experts would be spending too much time worrying about the implementation issues and easily lose track of the application domain as well as validation with the domain experts. Some end-user programming systems solve this problem by providing forms for non-programmers to write "before-after" rules [4, 95]. To write such rules, one specifies the criteria for applying the rule on the 'left-side' and the behavior to follow on the 'right-side'.

This thesis takes a domain independent, but scenario-writing specific approach. It has the advantages of the domain-independent, task support tools, without their disadvantages. Also, before-after rules were extended to increase their expressivity with temporal information.

1.5 Example scenarios

For an initial illustration of some scenarios, figure 1.2 contains three scenarios in a telephony domain. In this domain, objects or agents interact simultaneously within the context of an environment. Scenarios are only meant to contain a certain amount of information (this is of course subjective and dependent on the scenario writers' skills). For an example of what has been omitted in the example telephony scenarios, we do not see the division between the system and environment. This division is less important for conveying a contextualized representation of the domain, because both the system and environment are elaborated to the level of detail necessary.

In the telephony scenarios, I've placed myself in the situation of calling Lewis at his workplace. By placing myself in the scenario, I am setting the stage for a

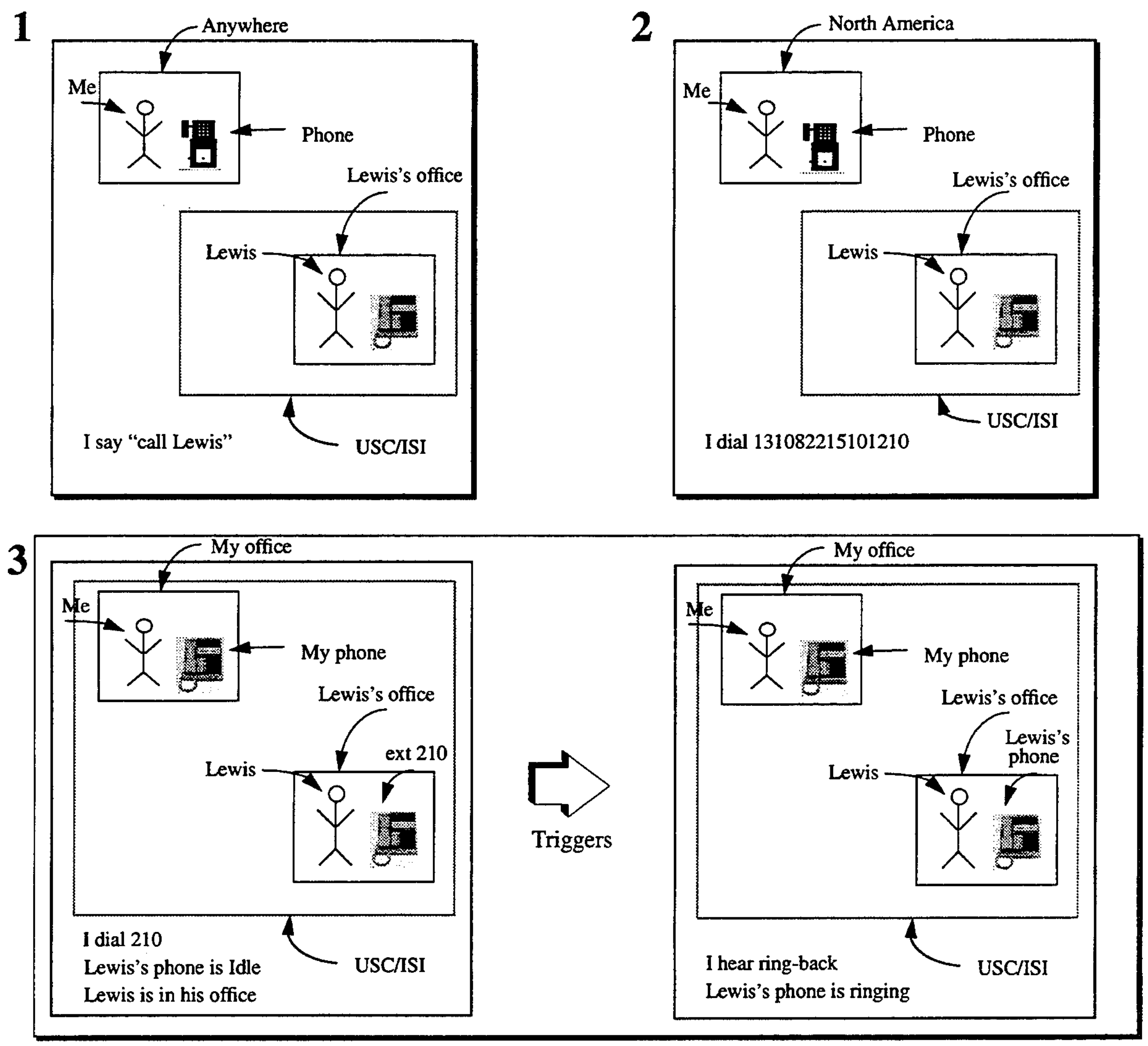


Figure 1.2: Telephony example

personalized, contextual view of the system. I am personalizing it to make it more detailed and more interesting by considering myself to be a participant (scenario immersion) as opposed to specifying some abstract role, such as "telephone customer." Alternatively, I could create elaborate characters with which to demonstrate usage [104]. Such concrete personification is helpful for people during envisaging.

Scenario 1 – This scenario shows a future goal. I would like to be able to pick-up a phone anywhere in the world and just say "call Lewis." The system would be able to recognize my voice and take the appropriate action.

Scenario 2 – This scenario depicts the current method of calling Lewis from a phone in North America. I just dial the area code, his phone number, and extension. With this simple scenario's partial context we can elaborate and discuss further details with a domain expert. The contextual nature of the scenario helps to evoke some questions to ask a domain expert such as: Do I need a country code when calling ISI from Toronto? Does the general rule about international calls apply? In telephony, service charges depend on the locations of caller and callee, so envisaging these spatial contexts is important. Of course, new mobile services are rapidly changing the way service is charged.

Scenario 3 – In the first *frame* of this scenario, I've dialed Lewis from my office when his phone is idle. The second frame describes a state following the first frame, namely, I hear ring-back and his phone is ringing. These were the result of my dialing Lewis's extension. This scenario could continue with, for example, Lewis answering his extension or my hanging up. Notice that the behavior is fragmentary and at different levels of abstraction, for example "dial 210" is a sequence of temporal ordered actions, i.e. pressing the different buttons. This sequence differs from an actual state, i.e. Lewis's idle phone.

The three scenarios contain roughly the same objects, namely myself, Lewis, and two telephones. It is the spatial context and the details of behavior (temporal context) which differentiate the concepts conveyed. Time and space are important themes to convey via scenarios. They are fundamental to all interesting real world domains.[3]

Any scenario support in scenarios like these will need to be considered along two dimensions, between and within. Different scenarios will have *between*-scenario properties. For example, the differences between what is stated in scenario one and two are different regions, like "anywhere" versus "North America" and differing activities, like saying and dialing. In scenario three the scenario has two frames and the behavior described is *within*-scenario. The focus is on the behavior, such as the details about what has occurred leading to the triggering of what will occur. This between frame behavior can be expressed in the form of the before-after rules.

In general, scenarios have a wide variety of uses such as those illustrated in scenarios one and two: expressing a goal of some future system or just describing the current state of a system to others. People use scenarios to communicate effectively especially when they are addressing people with different backgrounds from themselves. People also use formal languages for communication effectiveness when they convey information to others in the same field. Formality adds precision to a description. Formal scenarios can help domain experts and software experts achieve communication effectiveness and precision.

The requirements for automated support for scenarios rest upon having a precise representation or language for describing scenarios. The representation should support envisaging. Tools make it easier for people to actively engage in the task by allowing them to easily compose and construct scenarios. This is true especially when people can be involved in scenario writing by directly creating and manipulating the domain concepts. Tools for scenarios can support such tasks as editing, sorting, and filtering of information.

[3]Programming languages have also been explained via notions of time and space [29].

1.6 Real world basis

As part of this dissertation, a study of communication between software experts and domain experts was needed to understand both what people actually convey via scenarios and what information software engineers need from these scenarios. Borrowing from methods of ethnography [83], an observational study was performed.

This study involved the videotaping of knowledge acquisition sessions between software experts and former fighter pilots. They were developing a training environment simulating air-combat. They wished to include automated pilots which behaved and reasoned like human pilots.

This project exemplifies real world multi-site group development and is a more complex domain than the ones reported in the design studies of individuals and groups described in section 1.3. The scenarios in this domain are concrete examples of combat tactics. One of the major outcomes of this study was the recognition that these scenario representations were found to be richer than scenarios in the software design literature.

1.7 Scope of domains

The application domains considered in this thesis include ones in which objects are involved in complex and ongoing interactions with their environment. In order to achieve domain independence, several domains were investigated with differing levels of access to domain expertise. The five studied were vehicle-traffic control, air-traffic control, telephony, intelligent forces and satellite control.

Vehicle traffic-control and air-traffic control were early sources of example scenarios. While there was some communication with domain experts, these domains were studied mostly by direct observation of vehicle-traffic control and of requirements specifications for air-traffic control. However, merely reading the specifications of the the FAA advanced automation system, was not sufficient for understanding how planes were "handed-off". Visiting an FAA control center was necessary for seeing how hand-offs are currently handled. The controllers readily explained this domain concept as a scenario.

Telephony is another domain accessible by artifact. The telecommunications industry has produced much of the related work in scenarios for object-oriented methods. Scenarios are very important in this domain since many telephony problems are highly contextual.

The Intelligent-Forces project [25, 101] (an air-combat domain) provided the opportunity to study and capture meetings between domain experts and software developers in the context of an actual project. Videotapes were taken of meetings in which rapport was established and scenarios of fighter-pilot decisions were used to allow the software engineers to develop an understanding of the application domain.

Finally, NASA's deep space network ground control provided another opportunity to evaluate my work in a domain not considered during the development of REBUS.

1.8 Summary

Requirements envisaging is the process of transforming informal descriptions of software requirements into the precise language necessary for designing a software system.

Domain experts frequently express their requirements in terms of scenarios: partial descriptions of behavior in restricted situations. These scenarios frequently are not formally captured in the resulting requirements documentation. Thus, some, possibly essential information is lost in the translation of requirements between domain and software experts. As with the FAA's Advanced Automation System, this can result in costly and time consuming iterations of the requirements writing process.

This dissertation is a step in the automation of the requirements acquisition process. It provides a mutually-understandable, easily-learned, language, named REBUS, for communication between domain and software experts. REBUS enables the use of scenarios in this acquisition process, taking advantage of the way domain experts naturally express their requirements.

REBUS' design was guided by concepts found in a number of domains, including auto and air traffic control, telephony, and fighter pilot simulations. REBUS is evaluated in a new domain, NASA's deep space network ground control, which was not considered during REBUS' design process, thus providing a formative evaluation of REBUS's ability to facilitate requirements acquisition in a new domain.

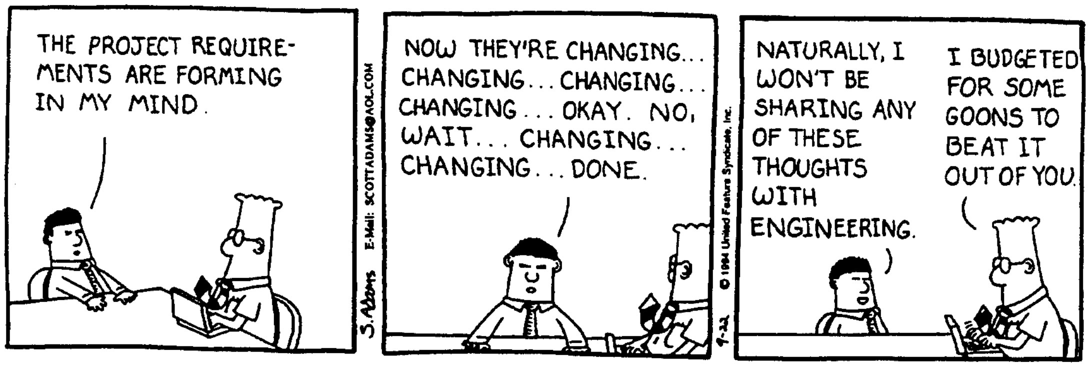
1.9 Thesis outline

Chapter 2 contains the desiderata of automated support for scenarios based on the ethnographic study in the intelligent forces domain. Chapter 3 contains further examples of IFOR scenarios, and compares them to existing scenario representations and automated support. Chapter 4 contains the REBUS language description. Chapter 5 contains the implementation of REBUS in an automated scenario capture tool called, SCtool. Chapter 6 contains the case-study performed to evaluate REBUS and SCtool. Chapter 7 contains conclusions and future work.

Chapter 2

Automated Support for Communication Using Scenarios

DILBERT reprinted by permission of UFS, Inc.


THE PROJECT REQUIREMENTS ARE FORMING IN MY MIND.
NOW THEY'RE CHANGING... CHANGING... CHANGING... CHANGING... OKAY. NO, WAIT... CHANGING... CHANGING... DONE.
NATURALLY, I WON'T BE SHARING ANY OF THESE THOUGHTS WITH ENGINEERING.
I BUDGETED FOR SOME GOONS TO BEAT IT OUT OF YOU.
S. Adams E-Mail: SCOTTADAMS@AOL.COM
© 1994 United Feature Syndicate, Inc.
9-23

2.1 Introduction

The central theme of this chapter is to identify the desiderata of automated support of scenarios. We are interested in providing support for effective communication between domain experts and software experts using automated tools. These people with different backgrounds and expertise need to communicate through time and space in order to understand and resolve problems. Scenarios are an important part of this communication and they should be captured in a persistent representational medium. So, we must examine what features scenarios must have so that, when incorporated in automated tools, effective communication can be facilitated.

In order to provide people with automated support for communication via scenarios, the scenarios need to be in a external representation that is domain-independent, evocative and precise. That is, for the representation to be effective for communication: (1) it must vividly evoke understanding from the scenario readers; (2) it must be eminently expressive for the scenario writers; (3) it must be readily manipulable and analyzable by people and automated tools with out a-priori domain knowledge.

This chapter is divided into a section on communication and a section which further addresses representational matters. Throughout, it contains examples from an observational study in the tactical air-combat domain.

2.2 Communication

Communication is used to transfer knowledge and understanding from one person to another [105]. Of particular concern is communication between people; between people and external representations; and finally between people and computers.[1]

2.2.1 Between people

Establishing good communication and collaboration between people is a complex process. One key problem that occurs when people with different backgrounds work together is that they need to establish a shared understanding of vocabulary. Such terminology will need to be expressible and precise within representations of scenarios.

To illustrate the problem of not having shared terminology, we first present dialog which comes from the observational study in the tactical air-combat domain. The participants can be grouped into pilots and software developers. The pilots were initiating the software developers in their domain at the request of the software developers. The software developers wished to ascertain the scope of the knowledge they needed of the application domain in order to develop a prototype simulation with which to demonstrate their software agent reasoning technology.

[1]Absent from this discussion is computer to computer communication.

The pilot has been asked to explain a long-range mission involving his plane and one enemy plane. He was asked to explain concepts from "first principles" and to describe his "plans." The dialogue was transcribed from the videotape of the meeting and it shows the pilot's attempt to incorporate the software expert's request for plans.

Pilot: *Your plan is... say for this intercept you want thirty degrees of target aspect when you fire the missile. You want to, at twelve and a half miles be on your reciprocal heading. You want to turn hard into the target at forty degrees. If you do that and you set yourself up at thirty thousand feet then you'll just turn hard, hard into it at forty degrees and then all of the things that change, the steering dots, the things that you're using to shoot a missile.*

The software experts request for "plans" was rather unsuccessful, since, to the pilots, a plan is a rather abstract term, and, to the software experts, a plan has a more precise meaning. The pilot might have used this term because the software developers and pilots are participating in a process of establishing "shared understandings." In order to develop shared understanding, people elaborate on ideas to a point where each participant believes she understands what the others mean [17]. Other activities for this process include: pointing to objects, repeating what has been said in a different way, and asking questions. If people had appropriate automated support for scenarios, they could let other people directly manipulate the concepts found in the scenarios, so that they can progress by pointing to concrete scenarios of their questions or of alternative situations.

Abstract software domain terminology is pervasive in the following statement by a software developer who is expressing concern for what is being said by the pilot.

Software developer: *It seems to me that part of what we need is a place where the descriptions that you're giving can be factored into sort of reusable components. ... so that this maneuver is constructed out of these components. So this little thing that's different here... we get to reuse all these components but suddenly we've got some other component to plug in. I'm not sure how to get that idea. It's not all* *the steps in the plan, that's not what we're decomposing, it's the pieces of knowledge that go into determining the steps.*

The software developer is still engaging in activity for shared understanding by gesturing to elements of the pilot's description, but uses terminology more natural to the software domain. It is a lengthy process to come to shared understandings. What is needed is a domain independent shared vocabulary which can serve as a more precise basis for communication and automated support. Such a vocabulary will need to include, for example, the concrete notions of measurement expressed by the pilots, such as "thirty degrees" and "thirty thousand feet." It should help people describe temporal and behavioral aspects in a more precise way, such as the exact conditions for firing a missile and the steps that follow, such as a 40 degree hard turn.

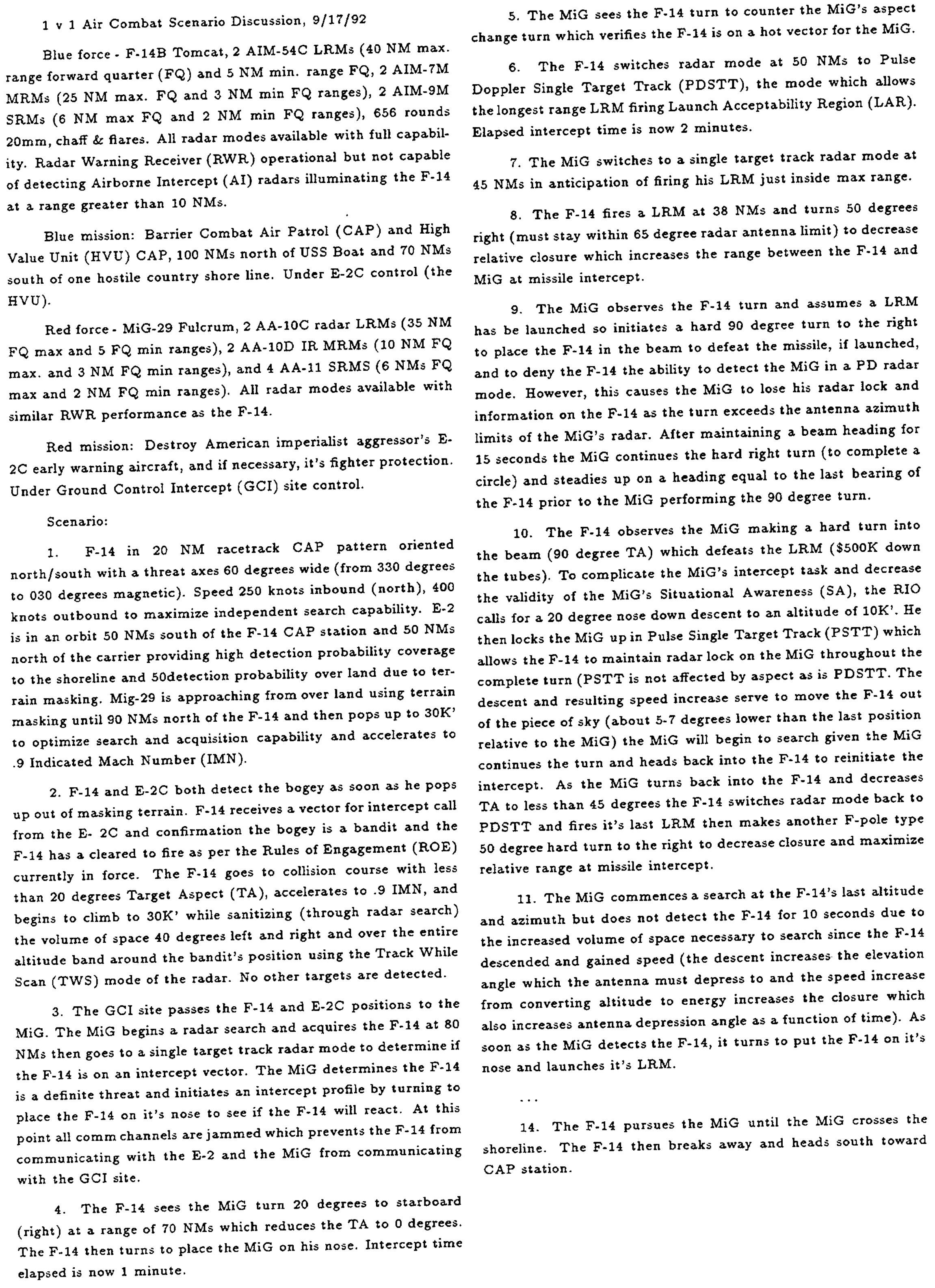
There is more to the communication and coordination process than establishing a shared vocabulary. When people are trying to engage in "win-win" negotiations [18, 9], they must also develop rapport. Advice for developing rapport includes such things as matching or mirroring other's physiology, vocal tone and tempo, and their choice of words [2]. Such behavior is not amenable to automated support.

After walking away from such a meeting, a participant might believe that she understands what the other is saying, but this may be a false sense of shared understanding. When it comes time for the action to be taken, or even when one returns to discussion in the follow-on meeting, it becomes obvious that not everything was understood. The details were not clear, forgotten, or never stated. Even *during* meetings, people have a limited capacity to understand and remember the others' terminology and knowledge. People need to return to concepts previously discussed for clarification. What helps people remember and validate knowledge from such meetings are persistent external representations. The next section will include a brief description of the external output of the meeting between the pilots and software developers; and a detailed examination of the interaction of people and external representations.

We note that during the meeting the communication process was dynamic, ill-structured, and spanned multiple media. Little beyond personal notes was captured during the meeting (except for the videotape which was used for this dissertation). Towards the end of the meeting, the developers asked the pilot to document via electronic mail the mission he presented so that it could be used as a basis for software development. That is, they requested the scenario which comprised nearly two hours of meeting discussion time. Figure 2.1 contains much of this scenario.[2]

The pilot has clearly envisaged the scenario and presented it in an organized manner. Although it misses some details found on the videotape, it does contain details not stated explicitly during the meeting. The first part of the text contains the initial situation, which includes rich details of the domain, such as the blue force (United States military force) - F-14B Tomcat, the missiles it is carrying, its radar modes, and its mission. Some details were omitted because of assumed context, an "F-14B Tomcat" is an American plane; and "LRMs" are long-range missiles, although the maximum and minimum ranges are given in detail and the radar modes are explained in further detail on the videotape. Some notions are not explained. For example, the pilot has switched viewpoints in step 2: What was the Mig-29 in step 1, is a "bogey" which is later confirmed to be a "bandit" in step 2. Such terminology needs to be captured in a manner that is manipulable, so that those who need further detail, such as the software developers, can find it. Such problems of detail and explicitness can be ameliorated with an appropriate shared external representation and automated tool.

[2]It has been edited with ellipses to fit on a single page.



1 v 1 Air Combat Scenario Discussion, 9/17/92

Blue force - F-14B Tomcat, 2 AIM-54C LRMs (40 NM max. range forward quarter (FQ) and 5 NM min. range FQ, 2 AIM-7M MRMs (25 NM max. FQ and 3 NM min FQ ranges), 2 AIM-9M SRMs (6 NM max FQ and 2 NM min FQ ranges), 656 rounds 20mm, chaff & flares. All radar modes available with full capability. Radar Warning Receiver (RWR) operational but not capable of detecting Airborne Intercept (AI) radars illuminating the F-14 at a range greater than 10 NMs.

Blue mission: Barrier Combat Air Patrol (CAP) and High Value Unit (HVU) CAP, 100 NMs north of USS Boat and 70 NMs south of one hostile country shore line. Under E-2C control (the HVU).

Red force - MiG-29 Fulcrum, 2 AA-10C radar LRMs (35 NM FQ max and 5 FQ min ranges), 2 AA-10D IR MRMs (10 NM FQ max. and 3 NM FQ min ranges), and 4 AA-11 SRMS (6 NMs FQ max and 2 NM FQ min ranges). All radar modes available with similar RWR performance as the F-14.

Red mission: Destroy American imperialist aggressor's E-2C early warning aircraft, and if necessary, it's fighter protection. Under Ground Control Intercept (GCI) site control.

Scenario:

1. F-14 in 20 NM racetrack CAP pattern oriented north/south with a threat axes 60 degrees wide (from 330 degrees to 030 degrees magnetic). Speed 250 knots inbound (north), 400 knots outbound to maximize independent search capability. E-2 is in an orbit 50 NMs south of the F-14 CAP station and 50 NMs north of the carrier providing high detection probability coverage to the shoreline and 50detection probability over land due to terrain masking. Mig-29 is approaching from over land using terrain masking until 90 NMs north of the F-14 and then pops up to 30K' to optimize search and acquisition capability and accelerates to .9 Indicated Mach Number (IMN).

2. F-14 and E-2C both detect the bogey as soon as he pops up out of masking terrain. F-14 receives a vector for intercept call from the E- 2C and confirmation the bogey is a bandit and the F-14 has a cleared to fire as per the Rules of Engagement (ROE) currently in force. The F-14 goes to collision course with less than 20 degrees Target Aspect (TA), accelerates to .9 IMN, and begins to climb to 30K' while sanitizing (through radar search) the volume of space 40 degrees left and right and over the entire altitude band around the bandit's position using the Track While Scan (TWS) mode of the radar. No other targets are detected.

3. The GCI site passes the F-14 and E-2C positions to the MiG. The MiG begins a radar search and acquires the F-14 at 80 NMs then goes to a single target track radar mode to determine if the F-14 is on an intercept vector. The MiG determines the F-14 is a definite threat and initiates an intercept profile by turning to place the F-14 on it's nose to see if the F-14 will react. At this point all comm channels are jammed which prevents the F-14 from communicating with the E-2 and the MiG from communicating with the GCI site.

4. The F-14 sees the MiG turn 20 degrees to starboard (right) at a range of 70 NMs which reduces the TA to 0 degrees. The F-14 then turns to place the MiG on his nose. Intercept time elapsed is now 1 minute.

5. The MiG sees the F-14 turn to counter the MiG's aspect change turn which verifies the F-14 is on a hot vector for the MiG.

6. The F-14 switches radar mode at 50 NMs to Pulse Doppler Single Target Track (PDSTT), the mode which allows the longest range LRM firing Launch Acceptability Region (LAR). Elapsed intercept time is now 2 minutes.

7. The MiG switches to a single target track radar mode at 45 NMs in anticipation of firing his LRM just inside max range.

8. The F-14 fires a LRM at 38 NMs and turns 50 degrees right (must stay within 65 degree radar antenna limit) to decrease relative closure which increases the range between the F-14 and MiG at missile intercept.

9. The MiG observes the F-14 turn and assumes a LRM has be launched so initiates a hard 90 degree turn to the right to place the F-14 in the beam to defeat the missile, if launched, and to deny the F-14 the ability to detect the MiG in a PD radar mode. However, this causes the MiG to lose his radar lock and information on the F-14 as the turn exceeds the antenna azimuth limits of the MiG's radar. After maintaining a beam heading for 15 seconds the MiG continues the hard right turn (to complete a circle) and steadies up on a heading equal to the last bearing of the F-14 prior to the MiG performing the 90 degree turn.

10. The F-14 observes the MiG making a hard turn into the beam (90 degree TA) which defeats the LRM ($500K down the tubes). To complicate the MiG's intercept task and decrease the validity of the MiG's Situational Awareness (SA), the RIO calls for a 20 degree nose down descent to an altitude of 10K'. He then locks the MiG up in Pulse Single Target Track (PSTT) which allows the F-14 to maintain radar lock on the MiG throughout the complete turn (PSTT is not affected by aspect as is PDSTT. The descent and resulting speed increase serve to move the F-14 out of the piece of sky (about 5-7 degrees lower than the last position relative to the MiG) the MiG will begin to search given the MiG continues the turn and heads back into the F-14 to reinitiate the intercept. As the MiG turns back into the F-14 and decreases TA to less than 45 degrees the F-14 switches radar mode back to PDSTT and fires it's last LRM then makes another F-pole type 50 degree hard turn to the right to decrease closure and maximize relative range at missile intercept.

11. The MiG commences a search at the F-14's last altitude and azimuth but does not detect the F-14 for 10 seconds due to the increased volume of space necessary to search since the F-14 descended and gained speed (the descent increases the elevation angle which the antenna must depress to and the speed increase from converting altitude to energy increases the closure which also increases antenna depression angle as a function of time). As soon as the MiG detects the F-14, it turns to put the F-14 on it's nose and launches it's LRM.

...

14. The F-14 pursues the MiG until the MiG crosses the shoreline. The F-14 then breaks away and heads south toward CAP station.

Figure 2.1: Air combat scenario written by a pilot

2.2.2 Between people and external representations

To enable the effective communication over time and space, persistent external representations must be used. These representations should be suitable to the task of scenario writing and manipulation between domain and software experts. In fact, when the building blocks[3] of the representation are sufficiently expressive the representation can aid the sharing of knowledge and elicit the asking of appropriate questions. Thus the representation helps to develop a precise record of shared understandings.

External representations play an important role in the social science frameworks of Activity Theory and Distributed Cognition [66]. In Activity Theory a key idea is the notion of *mediation* [58]. Nardi states [66], "Artifacts, such as instruments, signs, and machines mediate activity, and are created by people to control their own behavior. Artifacts carry with them a particular culture and history [58], and are persistent structures that stretch across activities through time and space." Scenarios need to be persistent because they are important for illustrating and explaining design decisions. Later, these decisions may change and may need to be traced back to the scenario.

Changing design decisions is a natural part of the process. During design, unplanned information from external representations can enter the focus of attention, trigger knowledge rules, and modify the designer's plans [36]. Guindon's study of systems analysts doing an elevator design task discusses the fact that domain specific depictions are an essential element of the external representations used by the analysts. Such depictions evoke human understanding and are an important part of scenarios in the air-combat domain.

Another aspect of communication that is important is the relationship between internal and external representations. A fundamental tenet of Distributed Cognition [28] is that problem solving behavior results from the interaction between external

[3]The primitive units used for modeling.

and internal (to each individual) representational structures. During the communication process, internal or tacit knowledge needs to be externalized as persistent, manipulable artifacts.

There are fundamental discussions in psychology over people's abilities to describe tacit knowledge. In *The Tacit Dimension*, Polanyi states [80]:

> I shall reconsider human knowledge by starting from the fact that *we can know more than we can tell.* This fact seems obvious enough; but it is not easy to say exactly what it means. Take an example. We know a person's face, and can recognize it among a thousand, indeed among a million. Yet we usually cannot tell how we recognize a face we know. So most of this knowledge cannot be put into words. But the police have recently introduced a method by which we can communicate much of this knowledge. They have made a large collection of pictures showing a variety of noses, mouths, and other features. From these the witness selects the particulars of the face he knows, and the pieces can be put together to form a reasonably good likeness of the face. This may suggest that we can communicate, after all, our knowledge of a physiognomy, provided we are given adequate means for expressing ourselves. But the application of the police method does not change the fact that previous to it we did know more than we could tell at the time. Moreover, we can use the police method only by knowing how to match the features we remember with those in the collection, and we cannot tell how we do this. This very act of communication displays a knowledge that we cannot tell.

Polyani's example illustrates three requirements towards bringing forth tacit knowledge.

1. The use of depictive external representations. The pictures showing noses, mouths and other features are external representations which are suitable for the activity. These pictures are variations on the basic building blocks (eyes,

a nose, etc...) of human faces. Some aspects of envisaging a domain are similar to face recognition by being highly depictive. Since scenarios convey behavior and other aspects of domain knowledge, symbolic and descriptive aspects of external representations are also important. The need for depiction and description will be described further in section 2.3.2.

2. The use of compositional building blocks. Manipulable, highly expressive languages start with a basic set of concepts. People compose them to represent and create other concepts. Such building blocks are not only the basis for supporting people with an adequate means for representing domain concepts, they are fundamental to the development of any automated support for scenario manipulation. One could not begin to develop a automated tool for such a collection of facial features without knowledge of the building blocks. A central tenet of this thesis is that scenarios have a common set of building blocks. During envisaging one needs to map domain concepts onto building blocks, and one needs the ability to compose and decompose such knowledge in a manner other people can understand. Building blocks will be further discussed in 2.3.4.

3. The act of actively doing: the witnesses and the police are actively engaged in an activity producing shared knowledge. The active and iterative engagement in a task by a domain expert is what Schön [90] calls "reflection-in-action." To support and evoke human understanding involves actively engaging domain and software experts in iterative, collaborative scenario writing.

Actively engaging in "reflection-in-action" is important to the requirements envisioning process. For example, in modeling the scenario presented in figure 2.1, a software expert focuses on the turning behavior of the F-14 in step 8. This example is an instance of a software developer leaving a meeting with a superficial understanding of a concept. It also illustrates the value of domain specific depiction for evoking tacit knowledge.

```
8.  The F-14 fires a LRM at 38 NMs and turns 50 degrees right (must
```

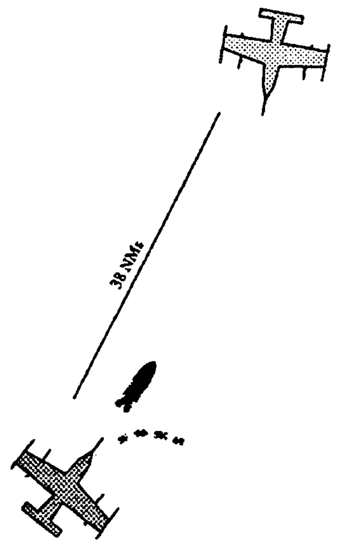


Figure 2.2: Depiction of situation in which F-pole is performed

```
stay within 65 degree radar antenna limit) to decrease relative
closure which increases the range between the F-14 and MiG at missile
intercept.
```

Note that the domain expert wrote step 8 in a cursory manner by stating that the pilot wants to decrease relative closure and increase range. The details of such knowledge are left unstated by the pilot.

In the case study, the software experts were responsible for further understanding and verifying the details of such tacit knowledge. The depiction in figure 2.2 illustrates the combat situation. The software expert understood, from the meeting, that the F-14 was performing an "F-pole" after firing the missile. One software expert thought that F-poles were performed to avoid debris. In drawing the depiction, she realizes that this knowledge was incomplete, since the F-pole was being performed for long range missiles. By asking additional questions, further reasons became apparent. What would happen if the MiG were firing at the F-14? The F-pole would help the F-14 avoid the MiG's missile range. What would happen if the missile misses? The F-pole would help the F-14 quickly turn around and fire. What would happen if there were already enough relative angle? Then further turning would not be required.

After engaging in such questioning, a definite description for an F-pole emerged. It is a composite of two simultaneous activities, that is, both a turn and missile support. The turn involves improving the fighter's position and velocity relative to the opponent and missile support involves making sure that the opponent stays within blue's radar volume at all times.

2.2.3 Between people and automated tools

To manipulate and analyze a representation, the choice of media is important. Automated tools provide added value for manipulating representations. The limits of the medium used to capture scenarios impacts further use of the scenarios. Instead of losing scenarios in the informal communication process, the goal is to capture scenarios in a manner which affords modification, maintenance, and analysis. It is generally difficult to modify, maintain, and analyze requirements for complex systems and the use of computer-based media is inevitable.

The air-combat scenarios observed in meetings spanned multiple media. The content of the communication was distributed across spoken, written (both text and drawings), gestural, as well as computational presentations [102] – all of which are perceivable and can be captured on videotape. The problem with videotape as a capture medium for scenarios is that it is unwieldy as an organizational structure. Techniques for searching and indexing are limited to linear visual search or require knowledge-engineered annotation (e.g. hyper-media). Videotape does not capture or structure the aspects of scenarios which are important without also capturing the meeting noise. For example, one can not easily search the videotape for situations in which a plane performs an F-pole or a plane performing an f-pole is discussed.

Consider using a drawing/text editor (one was used to create figure 2.2). Using such a tool we can manipulate text and graphics, but not F-14's or MiG's. At some point, the scenario writer will need to manipulate text and graphics to create the F-14's depiction, but this can be done in a more precise manner (e.g. the graphics could be specifically associated with the plane). The drawing/text editor doesn't support the task of editing F-14's, since the editor's view of the primitive building-blocks are structured graphics and characters. That is, the building blocks which the tool manipulates are at a different (lower) level of abstraction. Any tool which supports scenario writing will need to address such different levels of abstraction.

2.3 Representational matters

In developing automated support for a representation of scenarios, five dimensions are significant: 1) semantics: How can a representation have flexible semantics to support the scenario writers' conceptual intent and still support various domains. 2) graphical and textual presentation: Scenarios contain both, can one support both? 3) degree of structure: Scenarios can be written during ill-structured thought processes in order to uncover structure, so how much structure should a scenario representation provide? 4) building blocks: How many and what should they be? 5) and behavior. Scenarios contain behavior. How can it be represented?

2.3.1 Semantics

Semantics play an important role in human understanding and in determining automated support for a representation. For example, in the tactical air-combat domain, an "F-14B Tomcat" has precise meaning to the pilots in terms of the plane's characteristics. This meaning is vastly different to the software experts or to a computational system in which the formal meaning is only reflected in the encoding to the elements of a representation (e.g. as an object with attributes or as a string of characters.).

One basic way to avoid misunderstandings is to define terminology. So, for every term used between the domain and software experts there would be a item in the dictionary. This is problematic. These people with different backgrounds may not know how the terms relate to each other or to the physical world.

Consider three ways in which semantics can be embodied in a representation:[4]

1. **Conceptual categorization** is a basic approach with which to convey meaning. Concepts have meaning when they fit into one category as opposed to some other category. Categorization is one of the fundamental mechanisms for organizing knowledge. Lakoff [59] describes how categorization is a matter of human experience and imagination. Furthermore, he describes how some categories are "in the middle of the taxonomic hierarchy" and are learned first by children; who work up the hierarchy, generalizing, and down the hierarchy, specializing. The following hierarchy illustrates abstract to specific categories: object - animal - mammal - dog - beagle - Snoopy. Conceptual categories require the representation user to do some work. For example, the user must map domain concepts like "F14B Tomcat" onto a category provided by the representation, for instance, an "object" or furthermore, as a subcategory of another domain concept which has already been categorized.

2. **Analogy or metaphor** can be used to associate meaning with concepts. Some end-user simulation development environments have been designed and developed so people will map domain concepts to the environment's built in concepts. For example, Rehearsal World [27] uses a theater metaphor. In the case of an "F14B Tomcat" then a plane is a "performer" which moves around on a "stage". Another environment, ToonTalk^{TM} [53], uses cartoon character analogies. Table 2.1 shows the mapping between ToonTalk's concepts and computation abstractions. For example, a pilot would be encoded as a house, the control tower as a nest, and birds would be used to transfer messages between the pilot and the control tower.

3. **Domain specificity** is the final way considered to embody meaning. For example an "F14B Tomcat" would be a concept built into the representation.

[4]Not included in this discussion are various mathematical, formal languages due to the sophistication needed for their use.

ToonTalk^{TM}	Computational
city	computation
house	agent (or actor or process or object)
robots (with thought bubbles)	methods (or clauses or program fragments)
contents of thought bubble	method preconditions
actions taught to robot inside thought bubble	method actions
cubbies	tuples (or arrays or vectors or messages)
loaded trucks	agent spawning
bombs	agent termination
number pads, text pads, pictures	constants
birds	channel transmit capabilities
nest	channel receive capabilities
notebooks	program storage

Table 2.1: Mapping between ToonTalk^{TM} building blocks and computational abstractions from Kahn [53]. With the permission of Ken Kahn. © 1994 Ken Kahn

Thus, the term has meaning to the reader by a-priori definition. This is problematic when people have different backgrounds.

The domain specific approach appears to have the advantage because the user of the representation is skipping any mapping, but in the reality of building a tool to support such a representation, a mapping has been fixed a-priori and is no longer a flexible representation in the face of new knowledge. Domain specificity can also be considered to a degree. For example, instead of building in a "F-14 Tomcat" a more abstract domain concept like "fighter-plane" might be built in. This would introduce the need for mapping, as in the conceptual category approach. If we are willing to consider building in fighter-plane, what about the more abstract "plane." This alludes to the issue of what and how many concepts are built in. This issue will be discussed in section 2.3.4.

Analogy and metaphor approaches can be used as a very evocative part of the design process [62], but consider using them in the representation which one uses for making domain concepts precise. The metaphors in Rehearsal World and ToonTalk were both designed to let non-programmers (teachers and children, respectively) write programs in which objects interact in a simulated microworld. But the use of metaphor and analogy can conflict with the application domain. It can fail as a means to achieve precise communication, since built in metaphors rely on concepts which can either conflict with the problem domain or provide the communicator with an unnatural vocabulary.

The conceptual categories of a representation have to be easy to understand and useful for the task, but still independent of the participants' background and experience. To help software experts understand the application domain, domain experts have to provide an encoding to the categories of a representation. Dvorak and Moher performed a study [24] in which programmers were asked "to design object-oriented class hierarchies based on lists of properties similar to those which might be extracted from a software project requirements document." They found that "Differences in domain experience resulted in qualitative differences in their approaches to the problems and substantially impacted inter-subject agreement on the structure of the resultant hierarchies." Thus, it is very important for domain experts to be involved in the mapping of domain concepts to categories.

Consider a hybrid approach of definition, categories, domain-specificity and metaphor. The most successful end-user programming environment, the spreadsheet, is a hybrid. It has the abstract conceptual categories of sheets with rows and columns of textual cells which are related by formulas. It uses these in combination to embody a spatial metaphor. The meaning of a particular row and column comes from definition. The reason spreadsheets only partially provide semantics from such categories is that domain specificity towards financial calculations restricts the representation.

With respect to avoiding communication problems and providing automated support for scenarios, a balance must be achieved between definition, conceptual categories, domain independence, and analogy and metaphor. The REBUS representation provides a domain-independent conceptual framework based on notions found in real-world scenarios, namely objects, units of measurement and types, time, space and behavior. We are interested in the framework serving as a basis with which domain concepts are understood by the various scenario readers (including automated tools).

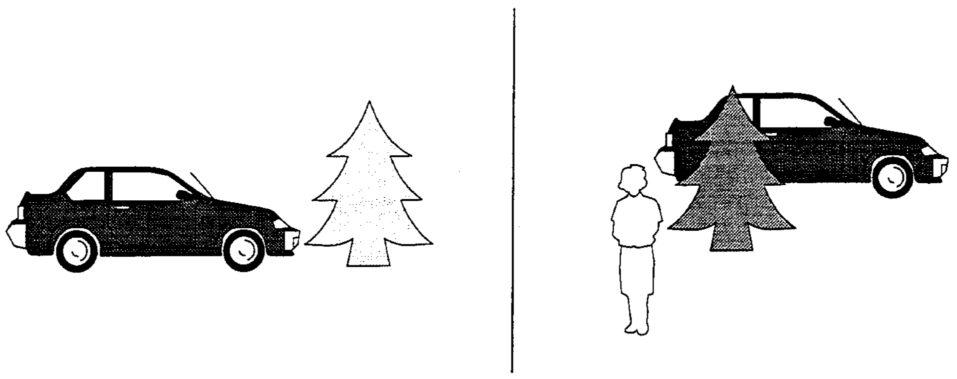


Figure 2.3: Two depictive interpretations of the sentence, "the tree in front of the car."

2.3.2 Graphical and textual presentation

Neither graphical nor textual presentations alone can serve as the ideal format for scenarios. Any generally useful (i.e. domain/task independent) external representation for scenarios must support graphical as well as textual presentations. Wurman states [105], "There is some consensus that pictures about concrete objects and events are understood more quickly, while words are favored when depth and clarity of comprehension are demanded, such as communicating abstract ideas. But this isn't enough to decide between the two. A rule that could be applied to information in general just doesn't exist. What has come through in many studies is that combinations of pictures and words are more effective than either alone."

With more than a few objects, depiction in the context of a spatial layout can aid precision compared to stating such information textually. A simple illustration of this is the natural language statement, "the tree in front of the car." Depictions (like the ones in figure 2.3) provide a more precise view of the various valid spatial configurations between the tree, car, and viewer. Although text can provide further precision in the spatial information, such as the distance between the tree and the car (e.g. five meters).

In scenarios in which multiple objects are interacting with their environment, graphical depiction is common in the external representations of domain knowledge. For example, figure 2.4 contains a page in the middle of a tactical air-combat scenario written by the pilots and shown to the software experts. There are graphical depictions of planes and textual descriptions of behaviors. In fact certain conceptual categories were frequently depicted, such as planes and spatial regions while others were generally conveyed with textual or spoken description. The simultaneous activity depicted in the context of this representation is used to convey behavior, in terms of spatial as well as temporal aspects of the domain.

2.3.3 Degree of structure

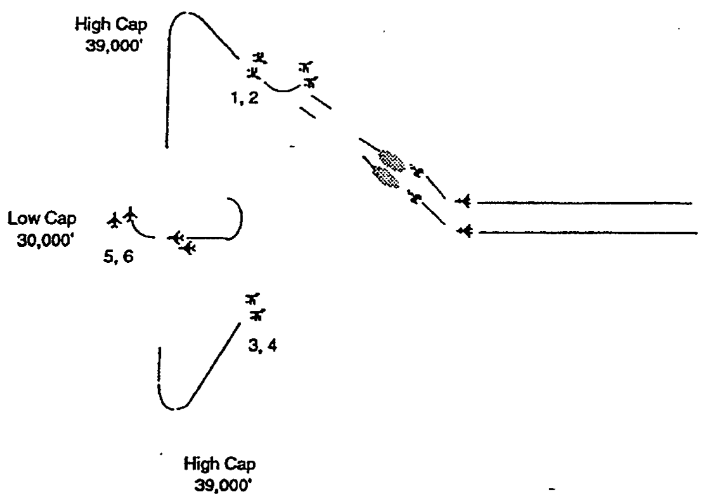
While support for graphical and textual descriptions is important, consider another cognitive dimension, the degree of structure or formality in a representation. Scenarios can be used during idea exploration (an ill-structured activity) as well as requirements validation (a more well-structured activity). So, an automated tool should be supportive of the intertwining of well-structured and ill-structured activities. Design involves iteration between well-structured and ill-structured processes [93]. To support the process, the scenario representational structure, must support different degrees of structure.

To illustrate different degrees of representational structure, consider four examples. First, consider a paint-by-the-numbers kit. This exemplifies a well-structured representation. Consider the other extreme, a blank page, as an unstructured representation. Finally, consider two intermediate degrees of representation structure: One, a child's coloring book with its scenes and characters already on the page; The other a coloring book [106] in which a pattern or doodle exists on the page (see figure 2.5). To use the latter, one envisions concepts (like one envisions patterns in clouds) and uses markers to bring forth the vision.

For further motivation to provide support for different degrees of structure, Goel's [30] work more formally defines the properties of ill-structured and well-structured representations and processes. In the context of ill-structured problem solving (e.g.

"Bagdad Taxi Drill" # 9b2b

Assumptions:
Enemy fighters laucnh counter fire at 1 & 2



t = 4:45 minutes

High Level Task Analysis for GCI Weapons Controller

25 Direct first element (1&2) to execute a hard 180° turn away from the second element (3 & 4)
26. Direct second element (3 & 4) to launch weapons on acquisition of IR target and r < z nm
27. Direct third element (5 & 6) to turn to intercept heading for enemy fighters (approximatey 180° in direction of enemy).
28. Monitor enemy fighters and assess reaction

Vector: Compute heading, velocity, (climb/dive) to desired point (relative to target projected location)

Monior: Compare actual to planned; recompute if projected result exceeds n miles and/or m° and/or t seconds

Assess: assign a rationale to detected enemy maneuvers that exceed N° heading change, M °/sec turn rate, or significant change in speed

Vector 2nd group: based on timing ans/or spacing from first group according to planned tactic

Command Preplanned Maneuver: Based ontarget range and/or aspect; initiate maneuver; direction of turn; final heading; *degree of turn; vertical maneuver*
Command constant speed 4 g level turn
Command "hard" 5 g level turn
Command "weapons launch

Assess Enemy Response:(to maneuver) evaluate enemey state approximately 5-30 seconds after initiation of maneuver; may be trun toward, away, continue on course, or split formation, *climb or dive; accelerate*

Figure 2.4: One page from a tactic map

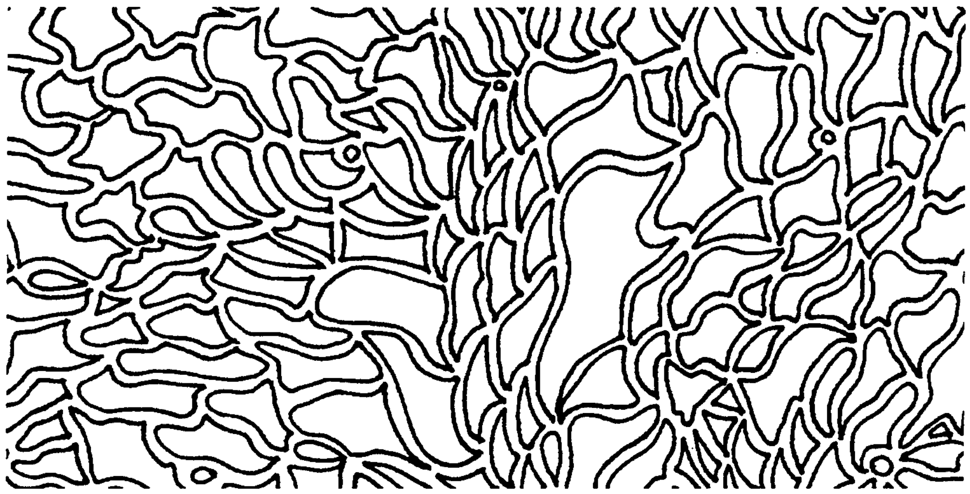


Figure 2.5: Example ill-structured coloring book. With permission of Karen Zand. © 1991 Karen Zand

graphic and industrial design), Goel's experiments, in which he restricted graphic designers to well-structured representations, indicate that ill-structured representations are needed by designers to facilitate the generation and exploration of alternatives.

One problem with having just a single level of structure, is that the resulting representation is less flexible for various tasks (note that it may be more supportive of a particular task). That is, what is well-structured and what is ill-structured is highly dependent on what is desired from the problem solving context. For example, to just capture the whiteboard sketches the fighter pilots made, one could provide automated support via a drawing program and pen-computer interface. The sketch can be stored in a well-structured representation (e.g. a stroke or pixmap) which can be further processed towards some forms of character/gesture recognition. The representation, via pictures alone, of all the possible spatial and temporal configurations of fighter-planes would require millions of different pictures. Without the ability to manipulate the level of abstraction, it would take a computationally infeasible amount of time to process all the pictures.

The user-interface of a scenario capture tool will have to support various degrees of structure in the scenario writing process. This can be accomplished with direct-manipulation graphical interfaces. The goal is to provide a representation which can support several degrees of structure. To some extent, this can be done by supporting building blocks and informal annotations.

2.3.4 Building blocks – neither too few nor too many

A scenario representation should have neither too few nor too many concepts. Consider a representation with too few building blocks, for example, Shlaer-Mellor's [92] object-oriented representation in which "everything is an object." Objects are the means with which to encapsulate behavior. One might consider objects to be a "minimal" set of building blocks, but the single-notion "object" representation system lacks rich semantics. The notion of an object is certainly compositional, but there is no conceptual framework with which to guide people. There are no distinctions in the representation with which to provide much in the way of semantic support for the domain or software experts.

Consider figure 2.6, the list of "things" which Shlaer and Mellor [92] recommend system analysts to look for as objects. This set of concepts is much larger than the set domain experts might intuitively consider as objects. As long as the modeling is non-intuitive to the domain experts, it will be difficult to get the domain experts involved. There needs to be enough semantic distinctions in the representation to guide the modeling.

One reason for such a minimal building block is to get a uniform representation for computer-based tools. Identifying everything as an object lets such tools support manipulation in a uniform manner, but the tools can't do much in terms of semantics. The semantics have to be distilled by the analysts when they decide on the objects. Objects are still a reasonable building block for a large set of domain concepts, and are useful in cases where the domain concept is not directly mapped to any of the other building blocks.

Having a larger set of building-blocks (as opposed to a single one), allows computers to handle a wider range of distinctions. Although too many building blocks may make it difficult to build and maintain a tool which supports these large numbers of building blocks. As long as the building blocks are primitive to a computer,

- *Tangible objects* are abstractions of the actual existence of some thing in the physical world.
  ▹ In a juice bottling plant: Pipe, Pump, Valve, Tank
  ▹ In a shipping application: Package, Delivery Vehicle

- *Roles* are abstractions of the purpose or assignment of a person, piece of equipment, or organization.
  ▹ In a university: Student, Instructor, Advisor
  ▹ In a chemical plant: Isolation Valve, Tank Inlet Valve
  ▹ In county government: Taxpayer, Jury Member, Voter

- *Incidents* are abstractions of some happening or occurrence.
  ▹ Accident (in a insurance application)
  ▹ Earthquake
  ▹ Election
  ▹ Delivery

- *Interactions* are objects that result from associations between other objects.
  ▹ Connection: the meeting of two pipes
  ▹ Contract: an agreement between two parties
  ▹ Intersection: the place where two or more streets meet

- *Specification objects* are used to represent rules, standards, or quality criteria (as opposed to the tangible object or role that meets these standards).
  ▹ A recipe represents the rules for making a certain quantity of a certain food (as opposed to the batch of food prepared according to the recipe).
  ▹ A compound represents the composition of a chemical (but not a particular sample of that compound).

Figure 2.6: "Objects" from Shlaer and Mellor [92]. Reprinted by permission of Prentice-Hall, Inc., Englewood Cliffs,NJ.

it is possible to provide more meaningful computer-based assistance and analysis than existing CASE tools can provide. For example, consider a CASE tool which supports data-flow diagrams. Data-flow diagrams have two building-blocks: nodes and edges. Such a tool can only check for consistency at the level of incoming and outgoing edges. Primitiveness is the significant feature needed to perform analysis.

One issue with larger numbers of building blocks is the possibly steeper learning curves to learn these blocks. The steepness of these curves depends, in part, on the the user's familiarity with the vocabulary and organization of the building blocks. This is the issue of domain-specific or domain-independent vocabularies discussed earlier. It is also dependent on the availability of good tools for finding and filtering the knowledge. Let's consider the Penman [63] and Cyc [35] representations used in the context of requirements engineering (which is not their developers' intended use).

Penman uses a large set of linguistic building blocks for organizing linguistic knowledge. The ARIES project [52] attempted to use Penman for requirements engineering in the domain of air-traffic control. Though many linguists consider it to be understandable and usable, the analysts who gained familiarity with Penman found its use difficult[50]. A smaller set of initial building blocks might provide more flexibility for organizing domain knowledge.

The use of Penman by linguists represents the case of providing a large domain specific vocabulary for use by domain experts. This approach is problematic for requirements engineering when it comes to communication between domain experts in the same field. Each may understand and use the vocabulary in slightly different ways in different contexts. Given a large vocabulary this approach may delay the process of uncovering misunderstandings.

Cyc provides a large number of building blocks. These building blocks are intended to be domain independent and to embody common sense knowledge about the world. The developers of Cyc believe that if they provide a representation with a rich set of well organized and layered building blocks then it could be used by others as a shared basis for developing knowledge-based systems. This effort has been underway for nearly ten years and while they have developed tools to let a knowledge engineer find and filter the knowledge, the current learning curve is several weeks. This seems too long for those engaged in requirements engineering.

To make its large representation more manageable, Cyc uses "microtheories" and "contexts" to organize and group the various building blocks. The solution provided for spatial concepts is thus,

> to use a number of globally inadequate but locally adequate theories of space. For example, we are working on (1) simple diagram-like representations, (2) computer-aided design-like representations that build solids and surfaces out of a small number of primitives, and (3) device-level representations that primarily deal with the topology of a device by using a number of primitive components and using a small number of ports for each primitive and a small number of ways in which two primitives can be connected. Although none of these abstractions is sufficient as a general approach to representing space, for any given problem, one of these (plus a few more that we are developing) is often adequate. These various abstractions of space are organized into a hierarchy because some are just refinements of others.

As they later state, there is the problem of determining "when to use which context, when a context is insufficient, when we need to enter a new context, and so on."

For REBUS, the small number of spatial concepts is based upon a small set of building blocks. These building blocks were derived from the study of scenarios which contained map-like sketches and from work in linguistics and cognitive science. REBUS users can group concepts into categories as necessary. This is further described in section 4.5.

Even the "right" number of building blocks requires some training to use. The goal is to try to make the set at natural as possible, perhaps enabling people to learn the tool in hours or minutes instead of weeks. Also using an appropriate set of building blocks opens the possibility of providing rich automated support for analysis of scenarios and of scenario collections.

2.3.5 Behavioral specification

Within a large body of computer science, a behavioral specification takes shape in abstract representations as data flow (e.g. data-flow diagrams, Petri nets) or control flow (e.g. state-transition diagrams, StateCharts). In either case, the formal computational system is in a single well defined state at any given time and transitions instantaneously change the system from one state to the next (i.e. a Turing Machine). In the real world, time has also passed.

For all but simple behaviors, these abstract representations of behavior are at odds with the notions used to describe behavior in a natural, fragmentary manner. Specifically, in scenarios, people express behavioral notions which encapsulate partial descriptions of state, time, and causality. In addition, people also need to express notions with concrete examples while having only partial or fragmentary knowledge of the overall behavior.

The high expressivity of formal languages for behavior still makes them good candidates for understanding what fragments of behavior are needed in scenarios for expressivity. StateCharts [41, 42] and Petri nets [85] are expressive formal representations with which we adopt some ideas for behavior fragments. A fragment that is useful from StateCharts is the historical state. Petri Nets have causal notions. That is, they support the expression of causal transition firings and they also have inhibitor arcs to restrain a firing. The goal is to provide a relatively small set of behavioral primitives that have semantics, and are understandable and expressive.

2.4 Chapter summary

Since scenarios are a means to facilitate communication between domain and software experts, our objective is to provide suitable tool support for scenarios in this complex communication process. This chapter focuses on explaining the desiderata of automated support for scenarios.

Throughout the chapter, examples are used from an observational study of domain experts (pilots) and software experts involved in the design of intelligent automated air-combat agents. These examples are used to convey aspects both of the requirements for automated support for scenarios and of the communication that takes place between domain and software experts.

This chapter began with a discussion of three types of communication: between people; between people and external representations; and between people and automated tools. The central issue in this discussion is achieving shared understanding between people and automated tools. Persistent and precise external representations are a means to achieve shared understanding. This discussion was a step toward understanding the strengths and limitations of external representations and automated support.

The chapter also establishes a set of target desiderata for a scenario representation and its automated support:

- The semantics of the representation should relate to the problem domain, without being domain specific. One cannot assume a tool has a-priori domain knowledge.

- The semantics should be understandable to people with different backgrounds. People will need some training to use any tool.

- The representation should support both depiction and description. Some concepts are best conveyed with depiction, while others require description.

- The representation needs to be flexible enough to support a range of structure. To support the design process the tool will need structured building blocks and less structured annotations.

- The representation should provide a set of building blocks with which domain knowledge is modeled. Building blocks are the basis for automated tools.

- The representation should have expressive behavioral constructs. A set of expressive notions for describing temporal and causal behavior are needed.

Chapter 3

Scenario Representations

There are only 5 ways of organizing information: by alphabet, category, time, magnitude, and location. Not 500, not 5000, but only 5. And it's the beginning place in communication. And in information display.

– *Richard Saul Wurman 1992, p.xxxv*

3.1 Introduction

It is important to consider the various scenario representations. A number of representations are relevant, but they do not meet all the desiderata described in chapter 2. This chapter begins by comparing a scenario representation from the air-combat domain to a scenario representation from the literature. Then, we present and critique the organization and encodings of various scenario representations found in the literature. Any automated support for these representations will also be described. Finally, we present a section which discusses the boundary between concepts found "within" scenarios (the focus of this work) and concepts found "between" scenarios.

3.2 A comparative

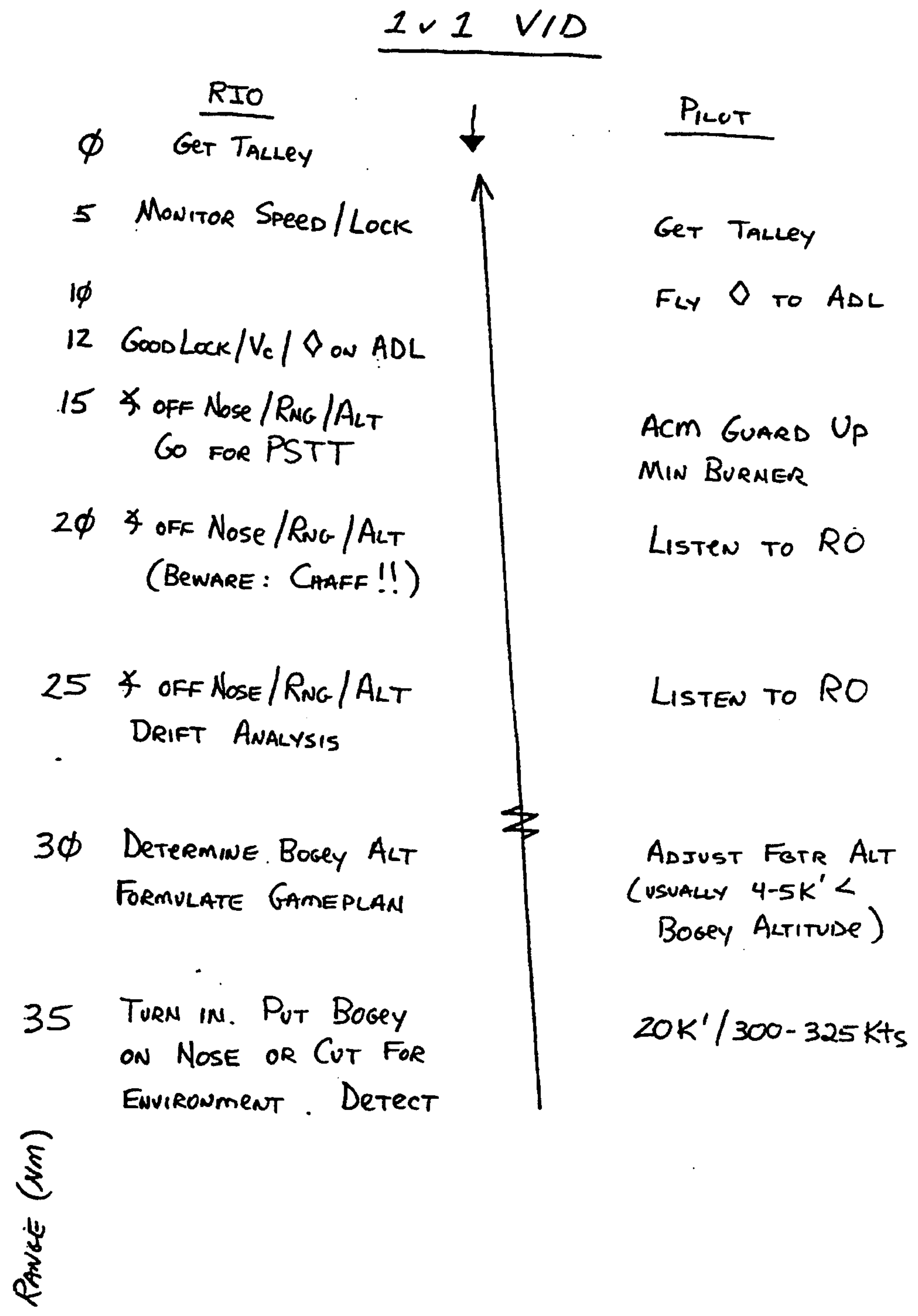
The concepts which naturally occur in scenarios written by domain experts can be compared to what is explicit in the building-blocks of the existing scenario representations. The "natural" scenario representations of the pilots contain rich semantic detail relative to the representations used in software design. Figure 3.1 contains a scenario from the air-combat observational study and figure 3.2 contains the elements of object interaction diagrams [49]. The two formats were chosen for comparison because they are pictorially similar.

The air-combat scenario in figure 3.1 was not designed for the software experts. In the observational study, it was presented during a meeting to show the communication between a pilot and a radar intercept officer (RIO). The diagram contains a mixture of parallel activities and interactions between the RIO and the pilot. It appears similar to an object interaction diagram, but appearances can be deceiving. For example:

- Notice the units on the left, they are in nautical miles, not time (although, this is still a temporal ordering). Spatial language was used to delineate the time intervals.
- In the videotape of the meeting, the pilot explained how this diagram is read, bottom to top.[1]
- The squiggly line through the long vertical arrow corresponds to the maneuvering for altitude.
- A third object (a bogey) is referred to by this diagram, but it is not readily perceived as important. That is, the bogey doesn't have its own column.
- Another object (the fighter-plane) and its attributes are left implicit.

As presented, the details are unclear to software experts or domain novices. Further detail should be represented explicitly. Without background various statements are unclear such as "20K'/300-325Kts" which is the plane's altitude and velocity. The K' stands for thousand feet and Kts stands for knots (nautical mile per hour). Such a diagram is not readily transcribed to an object interaction diagram without more collaboration between the software and domain experts.

[1]There was quite an audible reaction to this explanation. This seems to indicate that people expected to read the diagram top to bottom.



1 v 1 VID

Range (NM)	RIO	Pilot
Ø	Get Talley	
5	Monitor Speed / Lock	Get Talley
1Ø		Fly ◊ to ADL
12	Good Lock / Vc / ◊ on ADL	
15	∡ off Nose / Rng / Alt Go for PSTT	Acm Guard Up Min Burner
2Ø	∡ off Nose / Rng / Alt (Beware: Chaff !!)	Listen to RO
25	∡ off Nose / Rng / Alt Drift Analysis	Listen to RO
3Ø	Determine Bogey Alt Formulate Gameplan	Adjust Fgtr Alt (usually 4-5K' < Bogey Altitude)
35	Turn in. Put Bogey on Nose or Cut for Environment. Detect	20K' / 300-325 Kts

Figure 3.1: Interactions between a radar intercept officer and a pilot

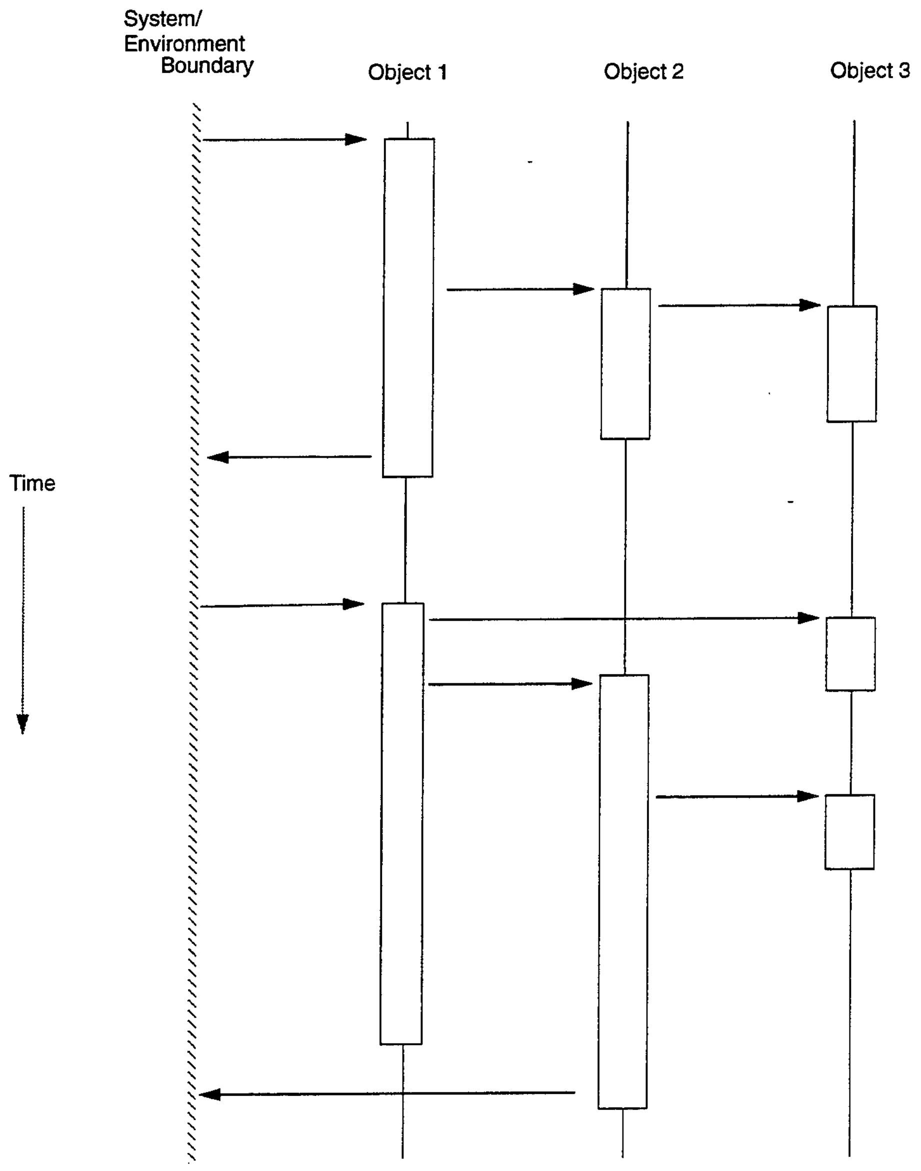


Figure 3.2: Elements of interaction diagrams

Object interaction diagrams are used in several software engineering methodologies, albeit under several names and variations, e.g. event trace diagrams, timing diagrams, message sequence charts [16, 49, 10, 32]. Such diagrams contain a temporal ordering of interactions (a.k.a stimuli or messages) between objects. As shown in figure 3.2, objects are positioned across the top, horizontally. The sequence of messages between the objects is read vertically (top to bottom). This sequence, generally corresponds to the advancement of time.[2] Other layouts (e.g. objects - top to bottom and time - left to right) are also used, but what is significant is the building blocks. In this notation, the building blocks are objects and messages.[3] Another explicit concept is the separation between the system and its environment.[4] Given only such building blocks, the tools for editing interaction diagrams do not meet all the desiderata and they do not explicitly support the rich semantic detail found in the scenarios written by the domain experts.

3.3 Survey of existing representations and automated support

In this section, examples of scenario representations are categorized by their prevailing visual structure: natural language, domain/task-specific depiction, tabular, formal specification languages, and diagrammatic notations. The examples come from a variety of sources and were used for different purposes. They illustrate various organizations and encodings of scenarios and the use of scenarios in various domains.

3.3.1 Natural language

Based on the desiderata both text and graphics are needed, but since the building blocks are different, this section will focus on text and the next section will focus on

[2]The exception, for example, can occur when the descriptive text on the left indicates a loop.
[3]A variant, with more building-blocks, is described in section 3.3.5.
[4]It's possible that such a separation would make the missing bogey and plane, more explicit.

First Jack is on-hook and Jack goes off-hook,
then Jack gets a dial-tone.

Next Jack dials Barry,
then Jack requests a connection to Barry
and Barry rings
and Jack hears ring-back.

Next Barry goes off-hook
then Jack gets connected to Barry.

Next Jack goes on-hook
then Jack gets disconnected from Barry.

Figure 3.3: Telephony scenario adapted from Kelly and Nonnenman [57]

graphics in the form of domain specific depiction. Note that they are both inadequate as the sole medium of communication and that they need to be supplemented with more structured notation. Natural language will be in a scenario representation. It is needed to meet the desideratum of support for varying degrees of structure.

The length of a textual scenario ranges from a couple of sentences to several pages. For longer scenarios, display technology imposes some limits on what can be seen simultaneously. An example of a natural language scenario from the telephony domain appears in figure 3.3. Examples from the air-combat domain are shown in chapter 2. From the standpoint of automated support, the building blocks are at the character, word, sentence, and paragraph levels. Thus, there are no explicit semantic constructs that automated tools can readily exploit (except for natural language recognition, which is discussed in this section) and that scenario readers can all assume are agreed upon and understood.

Organization can be used to impose structure on a natural language representation, as for example, in the paragraph numbering in the 1v1 combat scenario of figure 2.1. In this case, there is some temporal ordering conveyed by the numeric ordering of paragraphs, but the activities are still occurring in parallel. A scenario representation, like Karat and Bennett's [55] adds structure to the text that makes

Scenario Component	**Level of Detail**
Name	A short label used when referring to a scenario.
Situation Description	Running prose giving a concrete illustration of a situation.
Logical Essentials	With respect to the system, information that must be supplied in order to achieve the desired result within the system. With respect to the user, the representations and actions that must be made available by the system to the user. Information at this level is intended to be implementation-independent, what would be needed regardless of methods used to achieve the result.
Generic Steps	The sequence of user steps (sometimes ordered) that must be performed regardless of implementation method.
Specific Steps	A particular design will presume a series of user steps with particular devices and with system feedback to the user as each step is taken. Error analysis (what happens if a user makes a misstep or if information needed by the system is missing) can also be considered at this level.

Figure 3.4: Scenario Writing Guide from Karat and Bennett [55]. With the permission of Academic Press, Inc. © 1991 Academic Press, Inc.

up a natural language scenario. They use scenarios to describe user interfaces. In this context, they suggest that scenarios should contain the information described in figure 3.4. In the Objectory object-oriented method [49], there are natural language descriptions[5] and object interaction diagrams. The natural language descriptions have some structure. They include a name, a brief description, and a description of the scenario's basic course and alternative courses. These "courses" as well as Objectory's between-scenario relations are described in more detail in section 3.4.1.

Tools such as text editors and spelling checkers are relatively easy to use and they facilitate the manipulation of text. A more automated approach would be to use a natural language recognition system, but this approach has its limitations. For example, WATSON and KITSS [70, 71] were research projects aimed at providing automating tools to specify reactive telephone systems with scenarios (similar to the

[5]called "use cases"

one shown in figure 3.3). The recognition system required a-priori domain knowledge, but it was believed that this domain knowledge could be built in because the natural language used in writing the scripts was constrained enough to make automated understanding from telephony domain experts possible.

This belief turned out to be incorrect for several reasons. While the natural language was constrained enough for the *syntactic* aspects of the English used, the *semantic* aspects were wildly unconstrained. Sentence styles varied from simple and action-centered to elliptic, imprecise, inaccurate, subjunctive, and even metaphorical [39]. Another problem was that the natural language understanding techniques required a highly complete and virtually static built in domain model. Change was the rule rather than the exception and there was high overhead in maintaining the domain model [39]. Direct capture of natural language was abandoned as the approach and replaced by analyst transcription of natural language into a formal specification language (see section 3.3.4).

In REBUS, textual descriptions have structure and are explicit informal elements. That is, for example, explicit properties of the REBUS building-blocks include a name, a category, and a description. These are fields which are filled in by the scenario writer with characters, words, etc.

3.3.2 Domain specific depiction

In chapter 2 the need for domain specific depiction was discussed. The Simulacrum systems [11] developed at MCC were based on an empirical study of system analysts designing an elevator control system from a textual specification. The analysts were not domain experts. The study conducted at MCC and reported on in [37, 36, 38] as well as the storyboard layout of Simulacrum provided some motivation and inspiration for this work. Simulacrum explored a number of dimensions for design of a scenario acquisition tool: Domain dependence vs domain independence and WYSIATI (What you see is all there is) storyboarding.

The first editor, **Simulacrum-1** was a domain independent drawing program with storyboard sequencing support. It suffered from several problems which will

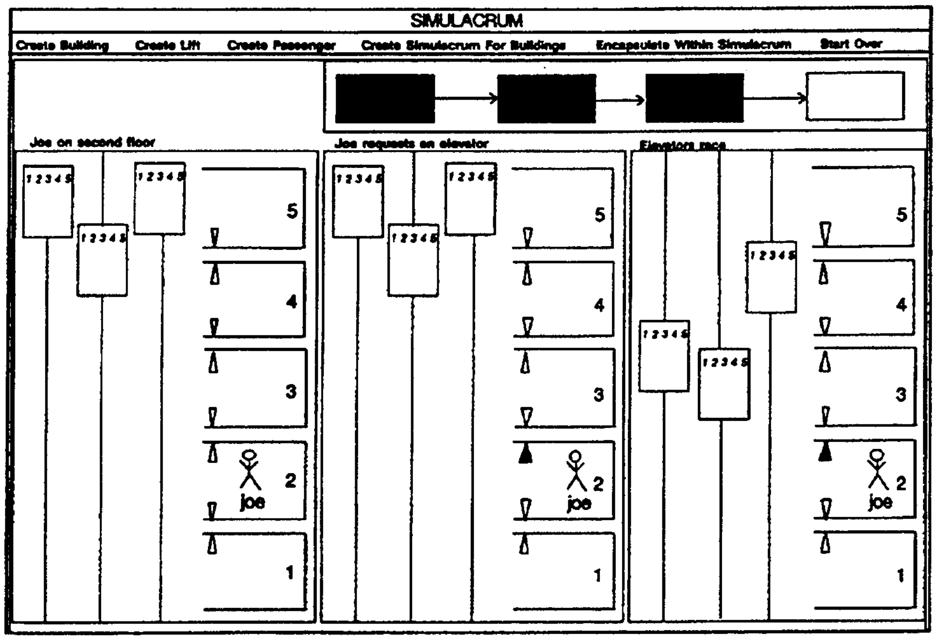


FIGURE 2. An elevator svstem clio.

Figure 3.5: Simulacrum-2 elevator scenario from Bridgeland [11]. With the permission of Plenum Publishing Corp. and David Bridgeland

arise in any system in which tokens are too low level and have weak semantics. The following discussion is based on [11].

1. The translation from the conceptual level to the graphic level was both time consuming and distracting. The sketch editor was domain independent, so objects manipulated by the editor were generic graphic things, such as lines, boxes, and lines of text. To make one conceptual change to a sketch, a user often had to perform many graphic operations.

2. The resulting storyboards often lacked conceptual integrity. A given conceptual relationship, for example, the notion that a elevator passenger was bound for a given floor, could be drawn in many different ways, and was. With support for a more symbolic approach to capturing concepts (instead of just graphics), one can at least encourage consistency through re-use of concepts across and within scenarios.

3. The relationship between the graphics and the intended semantics was not self-evident. There was no way to determine exactly what a given state sketch meant, except to query the person who had drawn it. This appears to be due to the WYSIATI restriction. With this restriction, there is just one level of structure with which to convey all the details. For example, the drawing of an elevator passenger visually does not expose all its attributes. What is clearly visible is just the current spatial relationship. This example can easily be handled in REBUS by a passenger *object* with an *attribute*, floor-bound-for.

Simulacrum-2 was built to correct the problems of Simulacrum-1, although the domain specific approach in Simulacrum-2 (see Figure 3.5) is not satisfactory. The semantics of objects in the lift problem were isolated and encoded, and direct manipulation presentations of those objects were built in. This constrained the editing, simplified the use, and restricted users to only plausible elevator system states. However this required considerable effort on the part of the analyst and over-constrained Simulacrum-2 to being a domain specific scenario editor.

REBUS alleviates the problems of Simulacrum-1 by providing more semantics on the low-level building blocks. REBUS does not include enough domain knowledge to restrict users to only plausible states (a task for the domain experts to model). An advantage of not building in domain concepts is greater flexibility and expressivity, while the disadvantage is the overhead required for the users to do the initial domain modeling. This modeling would have to be done in any case. REBUS provides a bridge to the gap between low level abstractions and domain specific concepts.

3.3.3 Table representations

Table based scenario representations (also called *scripts*), are an organizational variation of structured textual scenarios and object interaction diagrams. Some examples are shown in figures 3.6, 3.7, 3.8, and 3.9. The temporal order is explicit and read top to bottom (no loops). The table's column structure imposes some semantic support, namely for objects and their actions or responsibilities. Thus, there are

Agent	Action
Esther	Creates new meeting
Esther	Determines that Kenji is an important participant
Esther	Determines that Annie will be presenting
Esther	Determines that Colin is an ordinary participant
Esther	Types meeting description
Esther	Sets date range to be Mon-Fri next week (it's Wednesday p.m. now)
Esther	Determines drop-dead date is Friday noon
Scheduler	Sets time-out to be Fri 9am
Scheduler	Sends boilerplate message to Colin requesting constraints
Scheduler	Sends boilerplate message to Kenji requesting constraints and preferred location
Scheduler	Sends boilerplate message to Annie requesting constraints and equipment requirements
...	

Figure 3.6: Script - concrete version from Potts et al. [82]. With permission of IEEE. © 1994 IEEE

relatively few semantic concepts and there is no support for domain specific depiction. The tabular format also encourages the use of short, cryptic statements of scenario steps. Additional representation structure is needed to prevent confusion or misunderstanding of these steps.

Table based representations can support several degrees of representation structure. This is illustrated with figures 3.6 and 3.7. The first contains abstract objects and actions, while the latter contains specific and concrete examples. Even though table representations can support several degrees of structure, it is difficult to incorporate depictions or other supplementary information into the table. REBUS provides support for such depictions and information.

The basic table notation is not very expressive in terms of behavior. Figure 3.9 shows a conditional rule notation which was recently added to the tabular notation of the Object Behavior Analysis and Design (OBA/D) method. Thus, simple (single object) conditionals can be expressed. In REBUS, more complex rules can be expressed.

No.	Agent	Action
1	Initiator	Request meeting of a specific type, with meeting info. (e.g. agenda/purpose) and a date range
2	Scheduler	Add default (active/important) participants, etc.
3	Initiator	Determine 3 participants
4	Initiator	Identify 1 presenter as active participant
5	Initiator	Identify initiator's boss as important participant
6	Initiator	Send request for preferences
7	Scheduler	Send appropriate e-mail messages to participants (incl. additional requests to boss and presenter)
8	Ordinary participant	Respond with exclusion and preference set
...		

Figure 3.7: Script - abstract version from Potts et al. [82]. With permission of IEEE. © 1994 IEEE

Script Name: Modification.1.example
Author: Donna
Version: 1.0
Precondition: exists(Spreadsheet),displayed(Spreadsheet)
Postcondition: modified(Spreadsheet)
Trace: Core Activity–Modification

Initiator	Action	Participant	Responsibility
User	select D1	Spreadsheet	select a cell
User	type text NEW	D1	set content to text
User	set text style to bold	D1	set text style to bold
User	select A2	Spreadsheet	select a cell
User	type text NAME	A2	set content to text
	(repeated select and type text for example)	B2, C2, D2, A3 through A10	
User	select Row 2	Spreadsheet	select a row
...			

Figure 3.8: OBA/D Script based on Rubin and Goldberg [88]. With the permission of the ACM. © 1992 ACM

Initiator		**Action**	**Participant**	**Responsibility**
Customer		inserts ATM card	ATM	accept ATM card
Customer		enters PIN	ATM	read PIN
IF	"the PIN is valid"			
	ATM	validates PIN	PIN Validator	validate PIN
THEN	"permit transaction"			
	Customer	selects a transaction	ATM	perform transaction
ELSE	"deny transaction"			
	ATM	notifies invalid PIN	Customer	read notification
ATM		returns ATM card	Customer	takes ATM card

Figure 3.9: OBA/D Script with conditional rule from Rubin et al. [89]. With the permission of ParcPlace Systems, Inc.

3.3.4 Formal specification languages

Some formal specification languages have been used for scenarios. For example, see figures 3.10 and 3.11. Hall [40] is using his formalization to explore automated support for scenario generalization. Benner [6] uses a path expression language to provide analysts with a means of scoping and constraining a large formal specification of a system for the purposes of validation. Such languages are behaviorally expressive, but they do not meet the other desiderata.

3.3.5 Diagrammatic notations

There are two prevalent forms of diagrammatic scenarios: object interaction diagrams and ordered, concrete traces through other types of diagrams or graphs. A simple version of an object interaction diagrams was described in section 3.2.

Figure 3.12 shows the building blocks of a visual design language called VDL [32]. VDL is to be used by software experts throughout system development. Thus this notation has less resemblance to the real world and more resemblance to systems languages. Figure 3.13 shows a scenario in VDL notation. Compared to simple object interaction diagrams, this notation adds a creation and destruction notation, and iteration and if-then-else constructs. A VDL scenario should be about the size of a note-card and it contains a identifying scenario number, a list of authors, a date,

```
SCENARIO PH-1:
 INITIALIZATION-SEQUENCE:
   (initialize!)
   (make-user! ALICE (1 9))
   (make-user! BOB (9 3))
 SCENARIO-BODY:
   (offhook! BOB)
   (press! BOB 1)
   (press! BOB 9)
   (observe= (ringing? ALICE) true)

GENERALIZED SCENARIO PH-1-G:
 ASSUMING INITIALLY:
   (equal (mode ?user1) idle)
   (equal (mode ?user2) idle)
   (not (equal ?user1 ?user2))
   (equal none (ext->user (list ?button0)(ext-map)))
   (equal ?user2 (ext->user (list ?button0 ?button1 (ext-map)))
 BODY:
   (offhook! ?user1)
   (press! ?user1 ?button0)
   (press! ?user1 ?button1)
   (observe= (ringing? ?user2) true)
```

Figure 3.10: Telephony scenarios from Hall [40]. With permission of IEEE. © 1993 IEEE

A validation question:

```
scenario ensure-trigger-for-set-alarm()
   roles(a1:alarm, t1:time)
   := [alarm-time(a1, t1)];
     ring-alarm[a1 = alarm, t1 =current-time]
```

The revised validation question:

```
scenario ensure-trigger-for-set-alarm()
   roles(t1:time, t2:time, a1:alarm)
   := [alarm-time(a1, t1)];
     (ring-alarm[a1 = alarm, t1 =current-time] +
     [alarm-time(a1, t2)] ==> terminate-ignore)
```

Approximation scenarios:

```
scenario human-scenario-1()
   roles(a1:alarm)
   := ([alarm-time(a1,``07:00:00'')];
     ack-alarm [a1] precondition alarm-ringing(?))*

scenario clock-scenario()
   :=([current-time = ``06:00:00''];
     [current-time = ``07:00:00''];
     [current-time = ``08:00:00''])*
```

Figure 3.11: Path expression scenarios from Benner [5]. With permission of IEEE. © 1993 IEEE

a title, a "to do" list, and a grouping descriptor to identify the part of the model or architecture (e.g. "content model", "reference model", "content architecture") that is addressed by the scenario.

Interaction diagrams are just one way to organize a diagrammatic scenario. Some examples of the other types of diagrams or graphs are shown in figures 3.14, 3.15, and 3.16. Figure 3.16 shows that the building blocks of object's and messages can be organized in different ways. In the figure, the Booch object message diagram and the interaction diagram model the same scenario. The scenario begins with an object labeled "aFOO." A FOO is a "Forward Observation Officer", so in object-oriented parlance aFOO is an instance of FOO. The first thing aFoo does is to "1:create()" an object labeled "a Fire Mission Task." The Booch Method notation [10] affords the presentation of structural relationships (The "has" relationship is shown by the solid boxes on the line connecting the two objects) in the same diagram as the scenario.

Graph representations have a small set of simple primitives and can be manipulated with automated tools. For example, users would modify requirements by adding or deleting nodes and edges in the scenario. One problem is that the adding of a node such as "on-hook and calls forwarded" needs to occur eleven times in their example. Describing a scenario in which the phone rings four times and is then forwarded to a receptionist requires several more nodes and edges. Graph representations either have one level of structure or have nested structuring. This nested structuring is primarily useful for encouraging abstraction and supporting a top-down, structured process. We aim to support scenario writers during a more ill-structured process, so this is at odds with the top-down structured process. Thus, it does not meet the desiderata.

3.4 Multi-scenario notations

So far, the discussion has been focused on the content *within*-scenarios. As introduced in section 1.5, there are also *between*-scenario properties and relations. I will briefly describe between-scenario support because much of the automated support

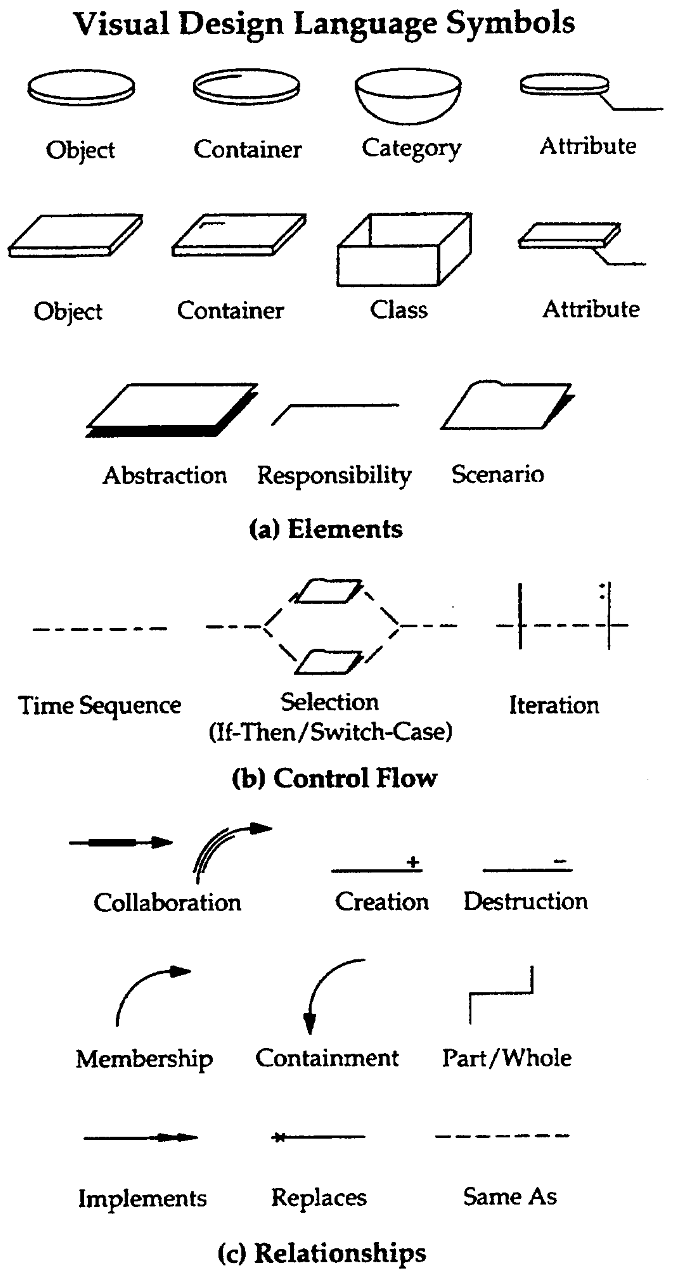


Figure 3.12: Visual design language (VDL) from Goldstein and Alger [32].[a]

[a]DEVELOPING OBJECT-ORIENTED SOFTWARE FOR THE MACINTOSH: ANALYSIS, DESIGN AND PROGRAMMING (pp. 137; 158), ©1992 by Neal Goldstein & Jeff Alger. Reprinted by permission of Addison-Wesley Publishing Company, Inc.

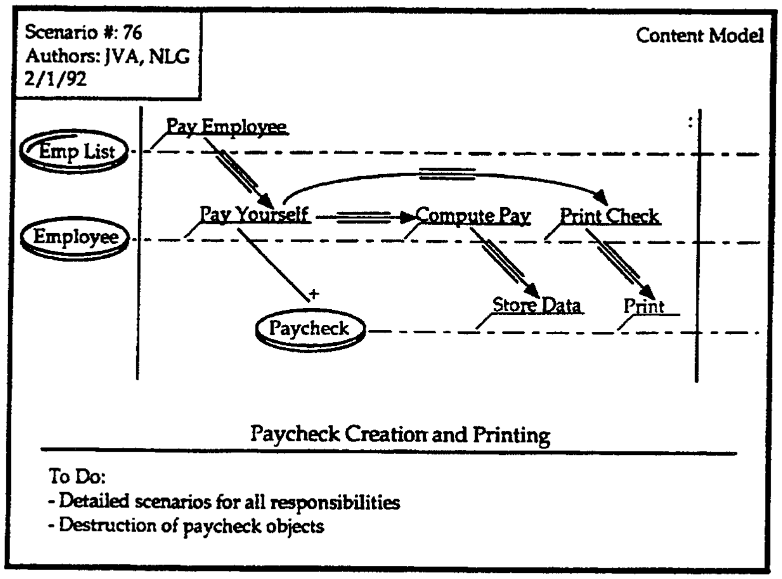


Figure 3.13: VDL payroll scenario from Goldstein and Alger [32].

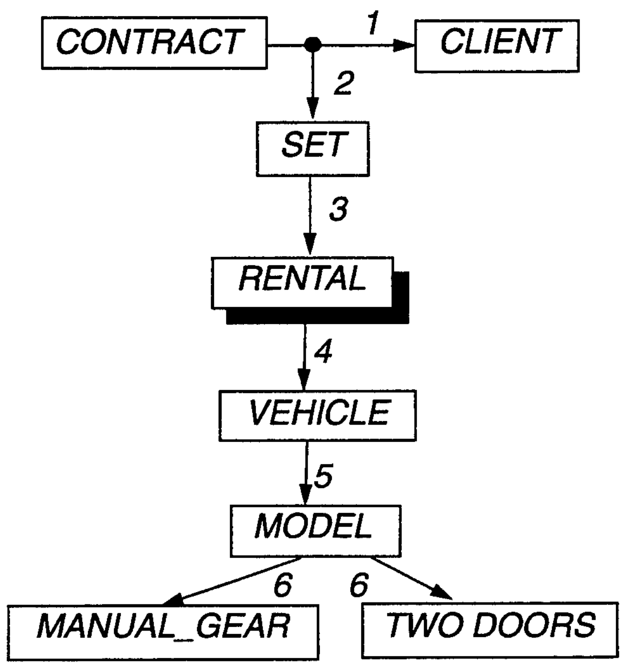
needed to manipulate and organize scenarios and thus facilitate scalability, is also needed to support traditional software development artifacts.

3.4.1 *Between*-scenario support

In order to support communication, scenarios should also be viewed within a larger context. Depending on the project, the number of scenarios can grow quite large. Scenarios will need to be grouped, archived, organized, composed, and decomposed (see figure 3.17).

Objectory [49] contains one possible set of relations over "use cases".[6] A *use case* refers to a *complete* course of events described in natural language between an actor and the system. For every complete course of events initiated by an actor, one use case is identified. The use case is normally divided into one *basic course*, which is the most important sequence of events, and several *alternative courses* which are variants of the basic course or errors that can occur.

[6]Jacobson et al. define a scenario as an *instance* (in the object-oriented sense of the word) of a use case. To clarify, the definition of scenario presented in chapter 1 encompasses both use cases and Jacobson's definition of a scenario.



A contract is prepared for a client:	1
Requested rented cars are grouped by client:	2,3
A rental plan is established for a specific vehicle:	4
A vehicle model is chosen:	5
Selected model is a two-door manual car:	6

Figure 3.14: Better object notation (BON) scenario based on Nerson [69]. With permission of ACM. © 1992 ACM

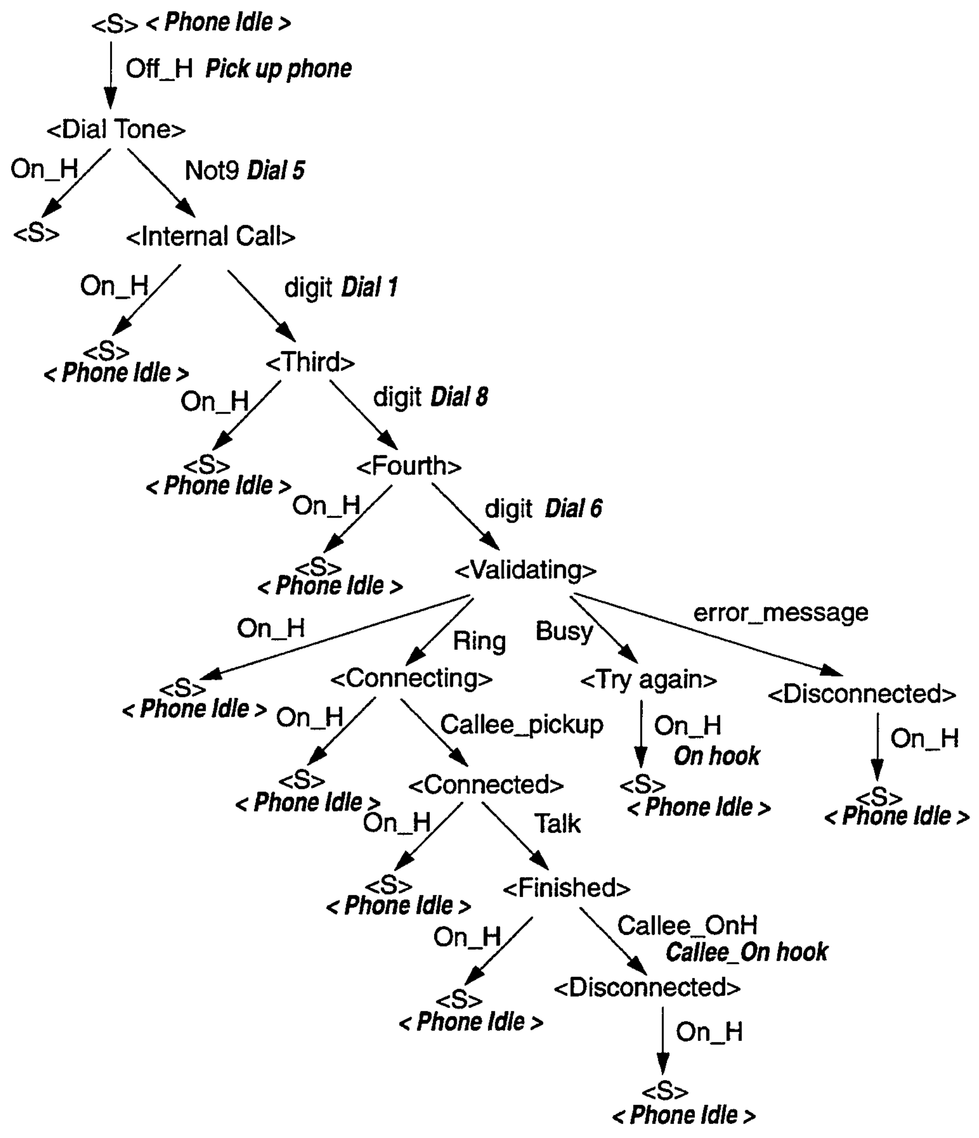


Figure 3.15: Scenario tree from Hsia et al. [44]. With permission of IEEE © 1994 IEEE

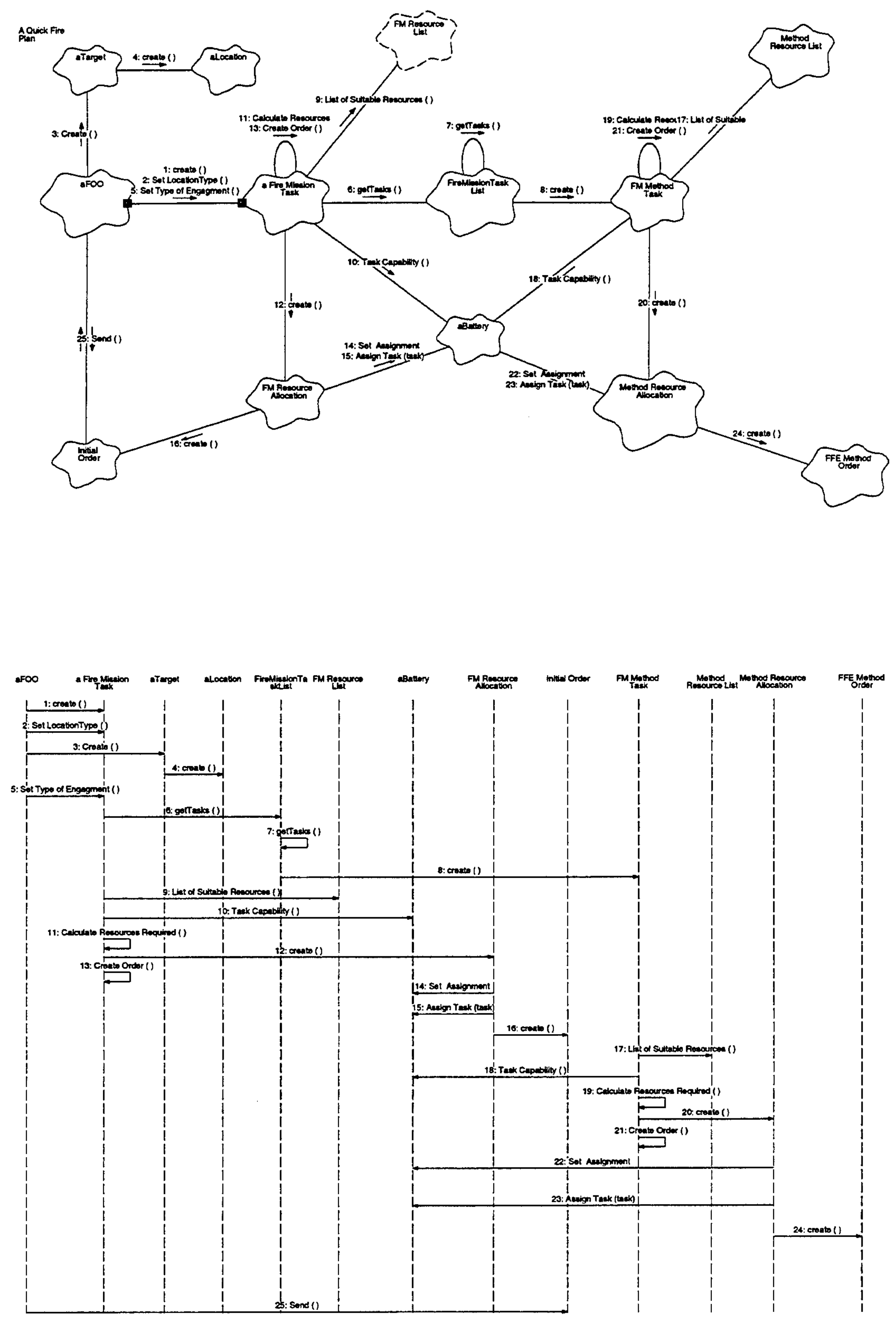


Figure 3.16: Booch object message diagram and corresponding interaction diagram. Diagrams courtesy of Captain Tony Marston, Canadian Forces.

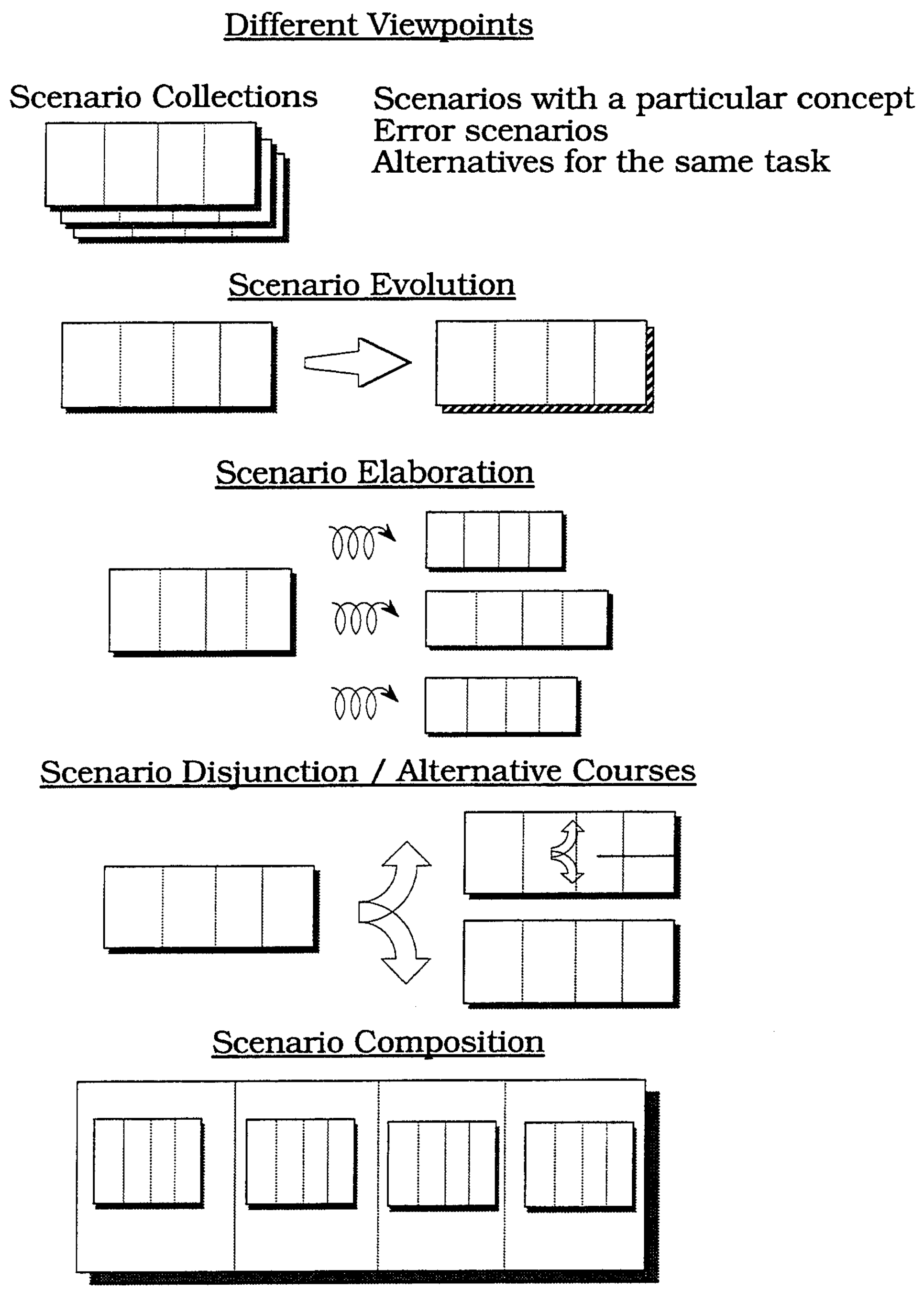
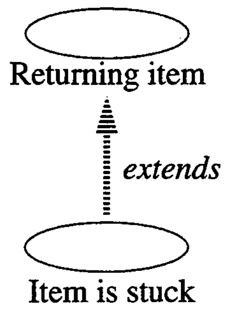
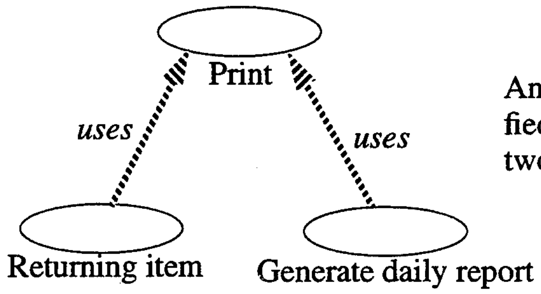


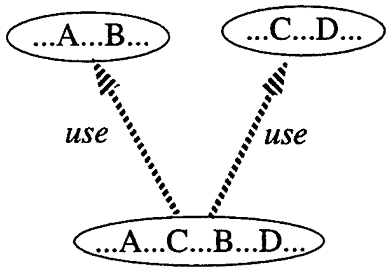
Figure 3.17: *Between* scenario considerations



The use case: <u>*Item is Stuck*</u> is inserted into *Returning Item* when *Customer* deposits an item that gets stuck in the recycling machine. *Operator* is called and *Customer* cannot turn in more items until *Operator* informs him that the machine can be used again.



An abstract use case *Print* has been identified to describe common parts between two other use cases.



A concrete use case uses two abstract use cases and decides explicitly how the interleaving is to take place.

Figure 3.18: Objectory - extends and uses relations [49]. With the permission of ACM. © 1992 ACM

The *extend* and *uses* relations are used to structure and relate use case descriptions (see figure 3.18). These can be viewed simply as textual inclusion mechanisms. The "uses" relation is used to reduce the maintenance on large sections of text that are used in two or more "concrete" use cases. Thus the "used" use case tends to be abstract, and doesn't have a complete sequence of events initiated by an actor. The "extends" relation can be used to simplify the description of a complex sequence of events by removing exceptional or additive behavior to an extending use case. Thus the "main" use case can be focused on the normal or basic sequence of events.

Alternative courses[7] are used to model related sequences which are part of the overall "class" definition for the use case. Depending on the sequence of events or other state information, an alternative course may or may not be executed. When the use of extends would reduce complexity, alternative courses are candidates for the extends relation. However, extends should not be used lightly. There is the notion of over extension which leads to a hard to maintain model - intolerant of change.

These relations are weak semantically since, for example, an explicit formalization of the behavior between the use cases does not exist. To illustrate this, notice that in the *Returning Item* use case (figure 3.18) the check for *Item is Stuck* does not merely extend *Returning Item.* The check must always occur. The events which follow from item is stuck could have been specified with a strong statement of restraint. That is, if the state of the system is "stuck" all customer behavior (relating to progress with the recycling machine) is prohibited, at least until the operator informs the customer the machine is working. Although further investigation is needed, the between use-case causality appears to be similar to the within scenario causal notions in REBUS.

3.4.2 Views and composite scenarios

Different users of a scenario capture tool will want their own views on a scenario collection. A developer may look for scenarios that contain a particular domain concept. Designers might organize them in terms of their design rationale, for example

[7]Alternative courses are *within* a use case.

in a graph which shows alternatives that can be taken to solve a problem. Analysts might organize them around a topic, such as failure scenarios. A domain expert might organize scenarios temporally like the tactic map in figure 7.1.

Supporting such views is not a central focus in my implementation. Others have developed tools to support archiving and the ability to query for a concept is the focus of databases. The most interesting form of support is reflected in the fighter-plane tactical map shown in figure 7.1. It contains temporal ordering, alternatives, disjunction as well as context. The notion of a scenario can be fuzzy in this case, i.e. is this figure also a scenario? We will call such things *composite scenarios.*

The question of whether a composite scenario is a scenario is one of nomenclature. The answer can depend on the readers' viewpoint, relative to the level of abstraction of system and environment behavior. For example, when all possible behavior is described in the composite, then it is no longer a scenario (its not a partial description). From the standpoint of simple scenarios, the need for a composite can be decided by the notable use of disjunction. The use of disjunction in scenarios is potentially problematic for the scenario reader because the scenario is not as concrete a trace of object interactions. Further investigation into when and how to present disjunction when conveying scenarios is needed.

3.5 Chapter summary

In this chapter, various within-scenario and between-scenario representations are compared and contrasted. This is done along several dimensions. First a "natural" scenario written by a pilot was compared with a common software engineering scenario representation, the object interaction diagram. Then, various scenario representations (natural language, diagrammatic, table based, etc) are explored for their strengths and weaknesses in terms of automated support and uses. Finally, the brief description and discussion of between scenario support. A richer set of between scenario support is described and also compared to a well known object-oriented methodology.

Chapter 4

The REBUS Building Blocks

Element – any of the four substances earth, air, fire, and water formerly believed to compose the physical universe. *Webster's Dictionary*

4.1 Introduction

This chapter defines the REBUS representation. REBUS is designed to satisfy the desiderata of chapter 2. REBUS is organized around a conceptual framework and building blocks. An overview appears in figure 4.1. The conceptual framework is composed of objects, measures and types, spatial elements, temporal elements, and behavioral elements. Except for objects, which are also building blocks, the last four items on the list are further divided into building blocks.[1] For example, spatial building blocks include regions, boundaries, and landmarks. In a particular domain, like vehicle traffic-control one would map domain concepts like roadway or lane onto region.

Parts of the REBUS representation are based on other work (e.g. linguistics, temporal logic, geographic information systems, expert systems, and algorithm animation). In addition to providing details on the building blocks and conceptual framework, each section contains a discussion of the similarities and differences between REBUS and other work.

[1]Note that throughout the representation, there are only 7$\pm$2 notions to choose from [64].

- ☐ Objects (concepts which have behavior)
    - o attributes, depictions, formula
- ☐ Measures/Types
    - o lists, units, named quantities, coordinate systems
    - o conversions
- ☐ Spatial Elements
    - o regions, boundaries, landmarks
    - o composites
    - o spatial relations
- ☐ Temporal Elements (behavioral sequence/actions)
    - o simple paths (single type, order, duration)
    - o composite paths
    - o temporal relations
- ☐ Behavioral Elements (causes, conditions, constraints)
    - o triggers (stimuli, constructors, destructors)
    - o restrainers (inhibitors, prohibitors)

Figure 4.1: Conceptual framework and building blocks

4.2 REBUS notation and terminology

First, some vocabulary common to the definition and explanation of REBUS.

4.2.1 Names and categories

A *name* is a descriptive sequence of characters. Names are the semantic labels of concepts reified in a building block. Acronyms tend not to make good names, but can be associated with a name.

A *category* is a name under which a concept is grouped or classified. Concepts have membership relations with their categories. So, concepts can belong to more than one category. The name of a concept is a category as well (e.g. car is-a car; WhiteBronco1 is-a WhiteBronco1 and a car).

Once a concept is named it is considered useful as a REBUS *type*. A type is the name or category of another building block. Note that type is also used in the framework terminology as one of the actual measure/type building blocks, but a distinction between the two isn't necessary since the measure/type building blocks have names and categories. Each element of the conceptual framework has its own set of category graphs, so concepts can be named and categorized according to their use without conflicting with other uses.

4.2.1.1 Discussion

Naming, categorizing, and describing are intrinsic to interacting with things and ideas. Naming has been studied in philosophy, linguistics, and psychology [14] and Carroll states [15], "Whatever underlies naming it is purposeful and achieves a reasonable compromise between competing and independent goals; including brevity, descriptiveness, inventiveness, differentiation, and complex social goals regarding self-presentation to and assessment of a communication partner." Names, categories, and descriptions are a necessary part of communication and modeling.

Names and categories are invented as needed to distinguish one concept from another or to denote their similarity. Sometimes naming mechanisms can be built into an automated tool, for example, spreadsheets. One of the reasons cited for the usability of spreadsheets is their built-in naming mechanisms for rows and columns (i.e. positive integers and the alphabet). One could provide default names for new building-block instances, but in the context of defining new domain knowledge, there is little evidence that naming can be done with automated support. In general one doesn't want to give non-programmers the task of naming things like variables [67], but most concepts already have names. One issue might be that domain novices will name and categorize concepts differently than domain experts, but it is important to encourage the naming activity in order to clarify one's terminology to others.

Carroll's instructions to the subjects in his naming experiments applies:

> Don't worry if at first you feel that you cannot distinguish notions like "categorizing" and "describing" from *naming* with absolute certainty. Naming implicitly involves categorizing and names are quite typically descriptive. However, the three can be distinguished: descriptions involve statements; they typically comprise clauses or even full sentences. Names, on the other hand, are smaller and syntactically simpler. They can be only nouns and noun phrases. Thus, "The cup has a chip missing." is a description, but "the chipped cup" can be a name for that cup. Names are usually very brief, rarely more than a few words long, often consisting of only one or two words.
>
> Categories, like names, are also brief. But a categorization like "cup" may not always sufficiently discriminate a to-be-named entity from others. "the cup" could be a good name for a cup, but only in particular circumstances (for example, if there is only one cup in the immediate environment). A name like "the apple juice cup" is descriptive, and it also categorizes, but it is name-like in that it suggests that there is a unique cup with the outstanding property of being used mostly for apple juice. If you are still quite uncertain about just what we mean by naming, discuss this with the experimenter before proceeding with the experiment.

In any case, when people are asked for a name for something, they tend to be able to invent one, albeit the name could be characterized by others as a bad name. If the name is bad, it can be changed. Automated support can be provided for changing names as well as propagating the changes.

4.2.2 Annotations

Annotations are visual or textual notes added to provide further comment or explanation. They are needed to meet the desiderata of support for a mixture of graphical and textual representations and support for varying degrees of structure.

A *description* is a textual place to leave comments about a concept.

A *depiction* contains domain or task specific graphics visualizing a concept. The depiction can be a sketch, a physical view of a real concept (from a scanned photo or video clip), a view of the graphical-user interface to a concept, a view of attributes, etc. To maintain strong semantics, a depiction or description is either tied to a building block or an explicit annotation.

An *attribute* is a name, type, value triple. Attributes are a more precise means to annotate a concept than descriptions and depictions. One can directly refer to the concepts attribute by name. For example, a concept (car) can have a attribute with name, color; type, car-color; and value, red. We use dot notation to refer to the car's color attribute (i.e. car.color). The attribute notion has more structure for the system than just having a description ("The car's color is red") or a depiction of a red car.

Attributes also have a convenient notation for collections. For example, a car dealership would have an attribute of type "car (collection)." The attributes value field would contain a comma separated list of cars (e.g. mustang57, taurus49) suitable for concepts needed in the scenario.

4.2.3 Mathematics

A *formula* is a computed function over attribute values. Formulas are analogous to spreadsheet formulas. Since there can be any number of mathematical functions in a domain, the list is not predetermined, but similar to some spreadsheet packages which provide mathematical functionality specific to particular financial domains. Mathematical functions include +,-,*,/,. . .

A *logical formula* returns a truth value. The operators are equals (=), less-than (<) , greater than (>), less-than or equals (<=), greater than or equals (>=), and ($\vee$) , or ($\wedge$), not equal ($\neq$).

4.2.3.1 Discussion

The mathematical aspects of domains were not emphasized in the domain expert scenarios. Domains with a great deal of mathematics, such as financial analysis already have a good deal of support in the form of spreadsheets or Mathematica^{TM}. Mathematical functions are still part of the domains considered, but they tend to play a secondary role in the scenarios. The mathematical libraries provided for use in automated tools still need to be easy to use.

4.3 Objects

An *Object* is the REBUS building block used to describe a concept which exhibits behavior and encapsulates state. Objects model domain-specific agents or components that have behavior. The differences between a REBUS object and an object oriented approach will be explained shortly.

An object is composed of a name, a category, a description, attributes, formulas, and depictions. Each of an object's attributes has a unique name, because it refers to a different aspect of the object. An attribute's value field is considered the default value. The value field may also contain the name of a temporally ordered sequence of values (i.e. a *path* which will be described in 4.6.3).

Here are some examples of objects in different domains:

Traffic-control domain: Objects include cars, traffic-lights, sensors,...

Fighter plane domain: A plane, a fighter plane, a missile, a F-14, a enemy plane (bogey), a friendly plane,...

Telephony domain: Telephones(attributes: receiver-status (on-hook, off-hook),...

REBUS objects are distinct from the object-oriented view of objects: There is no explicit abstract procedural interface specification (no list of methods). Objects encapsulate their depictions or views. Objects encapsulate formulas or constraints between attributes. There are more distinctions in REBUS, so fewer notions map to objects.

4.3.1 Discussion on modeling concepts

Some people [10, 34] believe that the problem of how objects and classes are obtained from the requirements document is the most difficult part of object oriented design. While some techniques have emerged, there is no agreed upon formal process for identifying objects and classes [24]. In Dvorak and Moher's study [24] of analysts constructing class hierarchies based on a specification, they found that differences in application domain expertise (not object-oriented design experience) resulted in qualitative differences in both process and product. Thus it is important to have domain experts involved in this modeling activity.

With automated support, modeling concepts as objects can be done in an incremental manner, based on the focus of the scenario. For example, in an initial scenario one of the attributes of a car object might be that it is a convertible. In considering a different scenario, the object named "convertible" is important, since the behavior of an significant attribute that is only valid for convertibles would be its top-status (i.e. the list type: top-up, top-down, top-moving).

In REBUS, spatial building blocks are used for spatial concepts. A car may also have an attribute which is its passenger compartment type. The passenger-compartment type is an example of a *region* building block which I describe in section 4.5.1 as a spatial building block.

4.4 Units of measurement and types

Weights and measures may be ranked among the necessaries of life to every individual of human society. They enter into the economical arrangements and daily concerns of every family. They are necessary to every occupation of human industry; to the distribution and security of every species of property; to every transaction of trade and commerce; to the labors of the husbandman; to the ingenuity of the artificer; to the studies of the philosopher; to the researches of the antiquarian, to the navigation of the mariner, and the marches of the soldier; to all the exchanges of peace, and all the operations of war. The knowledge of them, as in established use, is among the first elements of education, and is often learned by those who learn nothing else, not even to read and write. This knowledge is riveted in the memory by the habitual application of it to the employments of men throughout life.

John Quincy Adams in a Report to the Congress, 1821

In REBUS, there are five building blocks for measures and types: the unit type, the list type, a coordinate system, a named value or quantity, and conversions.

4.4.1 The unit type

The *unit type* encapsulates the notions of dimensions (e.g. length, time, and temperature) and units (e.g. 5 meters, 50 seconds, and 20 degrees Celsius), as well as mass nouns (e.g. 3 cars, and 7 jars of peanut butter). The unit type consists of a name, a category (dimension), and a shorthand view (a symbol or depiction) which has a display option of right side or left side (e.g. \$100, 50m). It also has a description, quantity restrictions, and component types.

Quantity restrictions limit the numeric values of a type, e.g. Real numbers between 0 and 1. An accuracy restriction is part of the quantity restriction e.g. accurate to two decimal places.

The unit type is one of the few building blocks where it is useful to provide a standard system, the international system of units (SI) [72]. SI contains the following 7 base units and 2 supplementary units:

- **Length** meter (m)
- **Time** second (s)
- **Electric Current** ampere (A)
- **Luminous Intensity** candela (cd)
- **Temperature** kelvin (K)
- **Mass** kilogram (kg)
- **Amount of substance** mole (mol)
- **Plane angle** radian (rad)
- **Solid angle** steradian (sr)

All other SI units are derived from the above, e.g.:

- **area** square meter (m^2)
- **volume** cubic meter (m^3)
- **frequency** hertz (Hz) cycle per second
- **speed** meter per second (m/s)
- **acceleration** meter per second per second (m/s^2)
- **electric potential** volt (V) $1 \text{ V} = 1 \text{ W/A}$
- **electric resistance** ohm (Ω) $1 \ \Omega = 1 \text{ V/A}$
- **force** newton (N) $1 \text{ N} = 1\text{kg}*\text{m/s}^2$

- **pressure** pascal (Pa) 1 Pa = 1 N/m^2
- **work and energy** joule (J) 1 J = 1 N*m
- **power** watt (W) 1 W = 1 J/s
- **concentration** mole per cubic meter (mol/m^3)

Other units are still expressible and the ability to add other units is needed. For example, the pilots use nautical miles as their unit of length.

4.4.2 List type

The *list type* is composed of a name, a category, a description, a optional comparison descriptor, and *elements*. Elements have names and descriptions. Elements can be made of named quantities which represent a value or range of values in a unit type or coordinate system. A comparison descriptor, if used, indicates that there is a total ordering between elements (e.g. short < medium < long). Currently, there doesn't appear to be a need for partial orders between elements.

Examples of list types:

telephony domain: `name`: Telephone-Receiver states, `elements`: on-hook, off-hook.

traffic domain: `name`: Traffic-Light colors, `elements`: red, amber, green.

fighter plane domain: `name`: Missile ranges, `elements`: short < medium < long.

The ability to create list types is important since they are domain-specific and more intuitive when precise values are not required. A problem with qualitative description is that mappings can mean different things in different situations within the same application domain. This can occur both by type (e.g. large could imply weight greater than 50,000 lbs, height greater than 20 meters, or fluid volume more than 8 oz.'s) or by value (e.g. large > 50,000 lbs and later on large > 5 lbs). This makes it important to examine list types across scenarios. Such examination may be unnecessary in acquiring individual scenarios, but becomes important if one wants to combine scenarios and to further clarify the situation.

4.4.3 Coordinate system

Coordinate systems are composed of a name, a category, a description, and one or more *axes*. An axis is similar to an attribute, but composed of a name, a type, and a description (instead of a value). An axis's ordering is imposed from its type. A specific point in a coordinate system is a list of values corresponding to each axis.

Examples of coordinate systems:

`name:` Cartesian coordinates `axes name` x `type` 0..1, `name` y `type` 0..1, `name` z `type` 0..1.

`name:` Color in RGB `axes name` Red value `type` red-value, `name` Green value, Blue value

`name:` Date `axes` Month, Day, Year.

Examples of points:

(January, 1, 1993), (0.3,0.2,1.0),...

Coordinate systems also need support for multiple textual presentation formats or depictions. For example, dates and times can be printed in several different ways, depending on prevailing custom. Of course, the different formats might be modeled as explicit categories.

4.4.4 Named quantities

A named quantity is composed of a name, value, type, category, and description, for example, Noon, Midnight, PI, BIGNUM, Origin (of screen coordinate system). One can even define terminology such as *now* where now is the smallest unit of time considered important (i.e. 1 second, 2 hours).

4.4.5 Conversions

While conversions are not apparent in the representations of scenarios, they are implicit. Conversions are important for understanding a domain and conversions should be explicitly acquired. The description of a conversion consists of a "From" type, a "To" type, a function, and a description.

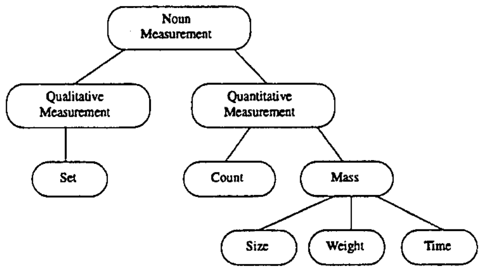


Figure 4.2: KRSL hierarchy of noun measurement types

4.4.6 Discussion of units and types

Units of measurement need to be explicitly supported as first class REBUS building blocks. They are very common in the natural scenario representations and they are also representative of knowledge which is not present when reading the existing software. In fact, in the intelligent forces project the units of measurement used for the fighter planes in a simulator developed at another site had to be re-discovered locally.

4.4.6.1 Other work on measurements/types

Other efforts at supporting units of measurement/types have produced or used rather complex vocabulary and highly detailed mathematical schemas. I will briefly describe three relevant efforts and compare and contrast them to REBUS.

The KRSL [48] hierarchy of measurement types is shown in figure 4.2. It is broken into qualitative and quantitative branches. The current effort to support units of measurement in KRSL is incomplete and it is not clear when this effort will be completed.

The Ontolingua knowledge sharing effort has defined "theories" of physical-quantities and standard-units [73, 74]. The authors developed an abstract schema which formalizes abstract properties at the expense of ease of understanding. For example, "Identity-dimension is the dimension of the so-called dimension-less quantities, including the real numbers." In REBUS, real numbers would be defined as a unit type and the notion of a dimension is captured as a category. Another example of the difference between REBUS and Ontolingua is the explicit notion of named quantities found in REBUS. Ontolingua organizes named quantities as subclasses of physical-quantities. For example, "zero-quantity" is a defined subclass of physical-quantity. In REBUS it would be directly captured as a named quantity.

Iscoe [47], in his thesis, provided similar measurement/types for domain modeling. In fact he devoted an entire chapter to the subject which maps down to the level of detail required to execute and validate type conversions. Iscoe's domains were transaction oriented business application domains and his schema is divided into scales (from mathematical measurement theory), units, quantities (fundamental and derived), granularity (from the physical sciences), population parameters (from statistics), and value set transitions. The REBUS list type corresponds to Iscoe's nominal and ordinal scales. The REBUS unit type roughly encapsulates interval and ratio scales combined with quantities and granularity. Population parameters which describe the distribution of values within a value set are currently not part of REBUS. Iscoe's work does not have the explicit named quantity or coordinate system notions.

4.5 Spatial elements

Landau and Jackendoff [60] present a fairly comprehensive account on spatial language. Although there is not a unified theory of spatial representation, they state:

> Understanding our representations of space requires invoking mental elements corresponding to *places* and *paths*, where places are generally understood as *regions* often occupied by *landmarks* or *reference objects.*

Objects (including oneself) are then located in these places. Paths are the routes along which one travels to get from place to place. These elements are likely to be critical in any complete theory of spatial representation.

Spatial representations are common in the pilot's scenarios, the spatial building blocks of REBUS are based on the concepts of maps. The primitive spatial concepts are: regions, boundaries, and landmarks [7].[2] For Landau and Jackendoff, boundaries are implicit in the notion of a region. Boundaries need to be used explicitly in order to formalize concepts like boundary crossings as critical components of the scenario. While the concept of a path is common in spatial language, in the REBUS vocabulary we consider it a temporal notion and describe it in section 4.6.3.

Maps used for scenarios are more sketch-like than those of cartographers (e.g. typical auto-club road maps), but they can still convey the spatial relationships between objects and their environment. In fact, for some domains, such as vehicle traffic control, spatial concepts are the significant elements of the environment considered in the design. Regions are the only primitive with spatial extent. This means that they are useful for representing containment. So, regions are the most common form of spatial concept. Other non-minimal variations on the spatial concepts include: composite region, point in region, and point on boundary. All of the spatial concepts reflect the elements which people refer to when gesturing or describing spatial concepts in front of whiteboards.

4.5.1 Regions, boundaries, landmarks

The region, boundary, and landmark building blocks are syntactically the same. They are composed of a name, a category, a description, a depiction, and an attribute list. They are semantically disjoint.

[2]The terminology used in [7] is areas, lines, and points.

4.5.2 Spatial composite

The spatial composite building block is included to support encapsulation of spatial concepts. A spatial composite has a name, a category, a description, a list of components, and a list of spatial relations (see 4.5.4) between components. For example, an intersection is composed of roadways, and roadways are composed of lanes. Intersections have boundaries such as center dividers.

4.5.3 Examples

Telephony domain: customer office, customer house, central office, telephone-pole, switching station,...
Traffic-control domain: Lanes, roads, left-turn lanes, intersections, approaching, in, and leaving regions, upstream,...
Fighter-plane domain: Land, sea, threat sector, shoreline,...

4.5.4 Spatial relations

Landau and Jackendoff [60] state "In addition to prepositions, there are many verbs that incorporate spatial relations; these can (almost invariably) be paraphrased by a simpler verb plus a preposition. For example, *enter* can be paraphrased by *go into*, *approach* by *go toward*, and *cross* by *go across*. Thus, the key element in the English expression of place is the preposition." They present table 4.1 as a fairly complete list of English prepositions.

The spatial relations and prepositions are highly context dependent and providing extremely precise notions is more difficult since one cannot assume that people with differing levels of spatial ability will be able to understand and use them [33]. One must still provide spatial prepositions since it is an area where visual depiction helps resolve communication issues. More precise solutions involve the use of coordinate systems, though supporting prepositions can delay that design decision. For example, depending on the frame of reference, there are three interpretations for the phrase, "The ball in front of the car" [86]. The intended frame of reference might

about	between	outside
above	betwixt	over
across	beyond	past
after	by	through
against	down	throughout
along	from	to
alongside	in	toward
amid(st)	inside	under
among(st)	into	underneath
around	near	up
at	nearby	upon
atop	off	via
behind	on	with
below	onto	within
beneath	opposite	without
beside	out	
Compounds		
far from	on top of	
in back of	to the left of	
in between	to the right of	
in front of	to the side of	
in line with		
Intransitive prepositions		
afterwards(s)	forward	right
apart	here	sideways
away	inward	south
back	left	there
backward	N-ward (e.g., homeward)	together
downstairs		upstairs
downward	north	upward
east	outward	west
Non-spatial prepositions		
ago	for	
as	like	
because of	of	
before	since	
despite	until	
during		

Table 4.1: Prepositions of English from [60]. Reprinted with permission of Cambridge University Press. © 1993 Cambridge University Press.

be the direction of the car, the direction the car is moving, or some outside observer perspective [86]. Each interpretation has a unique visual depiction. For many existing systems (especially 2-D graphical user-interfaces) the notion of bounding boxes is used for making spatial prepositions more explicit. Figure 4.3 contains the explicit REBUS spatial relations.

These relations have been organized into subsets based on the details needed to describe them. For example, a common way to consider spatial relationships between two concepts is by the alignment of their bounding boxes. To describe the relationship the names of the concepts as well as their offsets are needed. Since spatial concepts can have spatial attributes, relations between those attributes can be expressed with the logical relations: equals, less-than, greater-than, etc.

4.5.5 Discussion

Map concepts are also useful as the primitives for network depictions (i.e. nodes are regions and edges are boundaries). To talk about the spatial extent of an object, one can associate a region as an attribute of the object. If there is a question as to usage of an object or spatial building block to represent a domain concept, the decision should be made local to the concepts usage in the particular scenario. Context and behavior are the two distinctions to look for.

For the contextual distinction, Landau and Jackendoff state, "the standard linguistic representation of an object's place requires three elements: the object to be located (or *figure*), the reference object (called *ground* by Talmy), and their relationship. In the canonical English expression of an object's location, the figure and reference objects are encoded as noun phrases; the relationship is encoded as a spatial preposition that, with the reference object, defines a *region* in which the figure object is located. For example, in the sentence, "The cat is sitting on the mat," the figure (the cat) is located in the region described by the prepositional phrase *on the mat.* The region is in turn described by the reference object (the mat) and the spatial relation *on*, roughly, "contact with the surface of the reference object." For

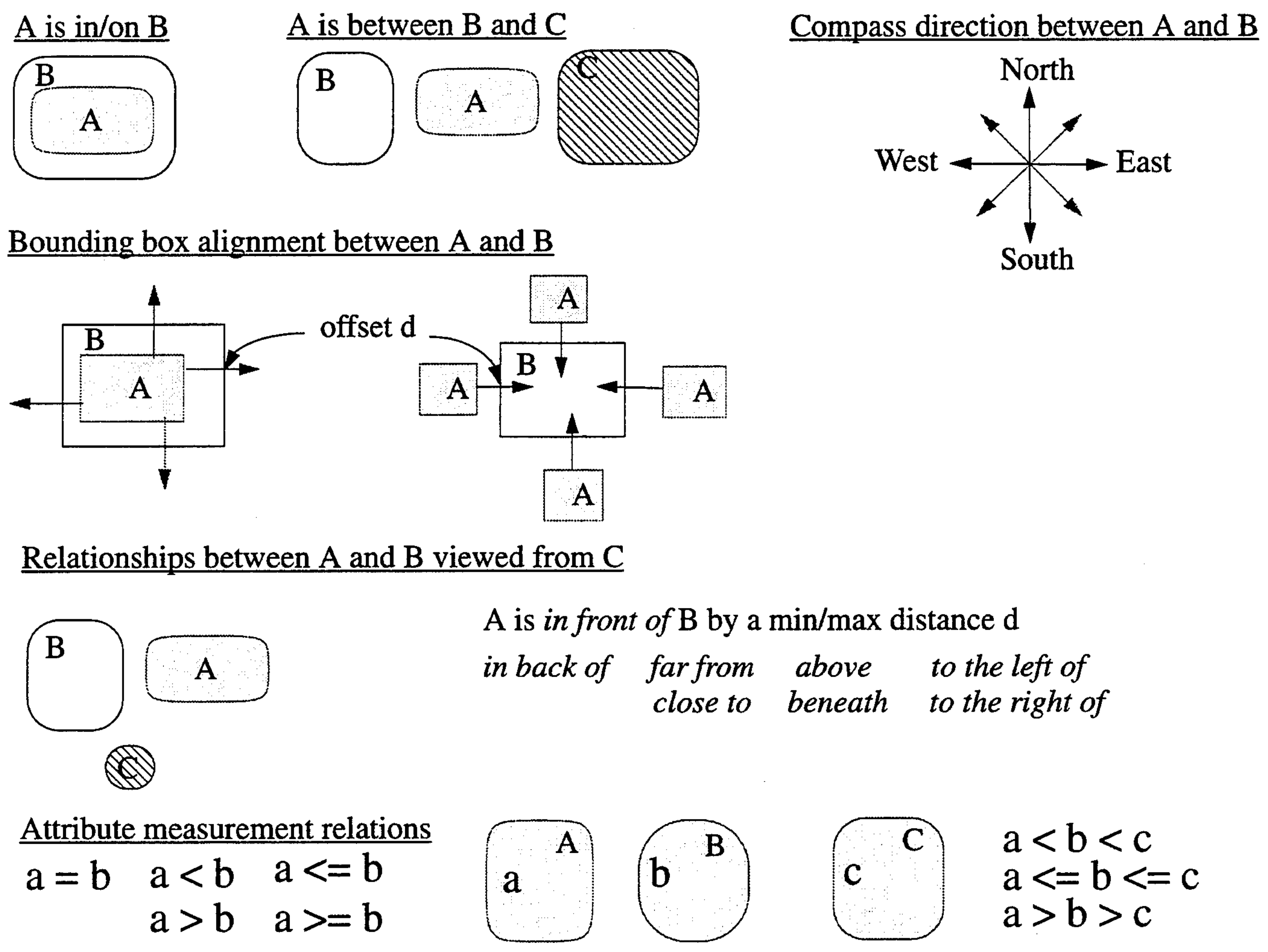


Figure 4.3: REBUS's spatial relations

the cat on mat situation, mat is a region since it merely serves as spatial context for the cat.

For the behavioral distinction, one relies on the notion that spatial concepts do not have behavior. This means that spatial concepts that have attributes that change over time should be modeled as objects. In the specification of a trigger or restrainer, one can still refer to objects contained in a region by including the region in the description.

Only some subsets of spatial relations can be recognized algorithmically. Douglas and Novick [23] describe an algorithm for determining a small set of relations (i.e. right of, left of, above, below, and between) from a picture, but the meaning of the use of the prepositions could become specific to a particular scenario or domain. What is occurring in the real world is that instead of natural language, coordinate systems (such as the global positioning system GPS) are being standardized to aid in pinpointing locations (to some degree of accuracy, e.g. 500 ft). Rimmer [87] provides a survey of qualitative spatial reasoning.

The set of spatial building blocks in REBUS is small compared to the Spatial Data Transfer Standard (SDTS) [103]. The SDTS currently has 200 entity types. Examples of these entity types include: airport, antenna, road, wall, beach, park.

4.6 Temporal elements

This section describes the temporal elements of REBUS. It begins with a definition of a *duration specification*, which is an encapsulation of duration information. Then, the set of temporal relations are described. Next, the basic temporal building block, a simple path, is defined. Paths provide an explicit means to describe temporal concepts. Paths have duration specifications and can be composed via temporal relations. These composite paths are also defined. We conclude with a discussion of paths: how they are used and how they relate to other work.

4.6.1 Duration specification

A duration specification is a template (not a building block)[3] consisting of a minimum duration, maximum duration, minimum begin, maximum begin, minimum end, and maximum end. Duration specifications are associated with a scenario, a path, or a path point (these terms will be defined shortly).

Generally the fields are filled in with temporal dimensions such as 1 Hour or named quantities such as Noon-Today or Now. Sometimes temporal durations are specified with other units of measurement or spatial concepts. For example, in describing a trip from New York to Los Angeles, minimum begin/end might be a landmark like the Statue of Liberty and maximum begin/end another landmark like the Hollywood Bowl.

Path points as well as paths (see section 4.6.3) can be thought of as 'intervals' or 'points' depending on the values in the duration specification. The usage semantically depends on the level of detail needed for the concept.

4.6.2 Temporal relations

Temporal relations are well known. They are specified by Allen's interval algebra [1] which provides thirteen relations between intervals (six are inverses). These are shown in figure 4.4. Allen's thirteen relations reflect a full characterization of the starting and ending of points for the intervals. These intervals are without duration specification. For REBUS, the set of relations between end-points is reflected in six ordering relations and five logical relations on durations, and a tolerance description (See Figure 4.5). One relation can describe, for example, that a combat mission finishes before a debriefing meeting starts with a maximum tolerance of eight hours. Compared to Allen's list, REBUS uses slightly different vocabulary and depictions for readability. Also, Allen's "equals" relation can be composed out of either A and B start together and finish together or A and B start together and their durations are equal.

[3]A duration specification is currently not a building block in REBUS because it is not separate from the building blocks which use it as a template.

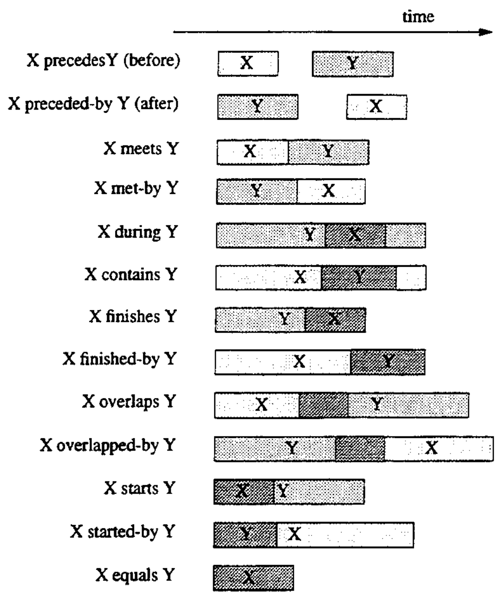
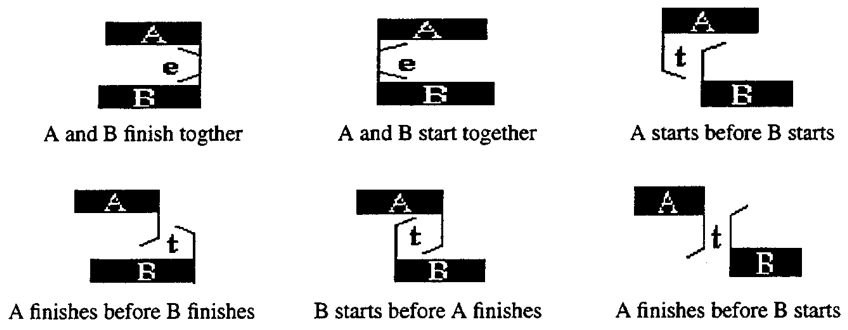


Figure 4.4: Allen's temporal relations

Order Relations


A
B
e
t
A and B finish togther
A and B start together
A starts before B starts
A finishes before B finishes
B starts before A finishes
A finishes before B starts

Duration Relations

Duration of A equals the duration of B	Dur(A) = Dur(B)
Duration of A is greater than the duration of B	Dur(A) > Dur(B)
Duration of A is greater than or equal to the duration of B	Dur(A) >= Dur(B)
Duration of A is less than the duration of B	Dur(A) < Dur(B)
Duration of A is less than or equal to the duration of B	Dur(A) <= Dur(B)

Figure 4.5: REBUS's temporal relations

Temporal relations can be logically combined, for example A and B start together *and* the duration of A is less than the duration of B. Currently, REBUS does not support conditional combinations, such as *if* A starts before B starts *or* B starts before A starts *then* A ends before B ends. Sets of temporal relations can be analyzed for consistency. (See [8] for research in temporal reasoning.)

4.6.3 Paths

Paths enable domain experts to express and encapsulate domain-specific sequences of behavior (activities). Simple paths consist of a name, a category, a description, an optional elaboration description, a point type, a duration specification, and path points. Path points are combinations of values of 'type' and duration specification. An *elaboration description* is a second form of categorization.

For example, consider a path named 'dial number' and category 'dialed number histories.' This path's points are '12138221511' and '18005551212'. This path could represent a history of phone numbers dialed. A second path, at a more detailed level of abstraction, could be named 'buttons pressed' and category, 'button pressed histories'. The paths have different categories. The elaboration-of field denotes a relationship between the two path 'types.' 'Buttons pressed' could be an elaboration-of 'dial number' and have the single digits of one of the phone numbers as its path points. The path, 'dial number' and 'buttons pressed' are composed of points typed as 'phone number' and 'button number,' respectively. A phone number is formed from a list of button numbers.

Simple paths are used to express a total ordering on an objects' attribute values. This means that in combination the duration specification of path points are restricted. Simple paths do not support the expression of temporal order indeterminacy. This can be conveyed with composite paths.

4.6.3.1 Composite paths

Composite paths are used to combine value sequences of multiple types and to specify temporal relations between concepts. A composite path has a name, a description, a category, an elaboration description, a list of component paths, and a list of temporal relations (defining the composition, see figure 4.5). The list of component paths is composed of paths which are referred to by their names and types.

4.6.3.2 Examples

Elevator domain: Simple path of elevator requests, (Floor 1, Floor 2, Floor 3, Floor 2)

Telephony domain: Simple path of telephone buttons pressed, (1, 3, 1, 0, 8, 2, 2, 1, 5, 1, 1). This could also be described as a composite path which would focus on the fact that different buttons are pressed.

Fighter-plane domain: A composite path: 1 G turn which combines velocity with turning.

4.6.4 Path discussion

4.6.4.1 Spatial paths

Spatial paths can convey spatial movement. If we require a spatial path, the path type would correspond to direction and the duration specification corresponds to distance. Spatial paths are likely to have a line depiction on whiteboard sketches (see figures 2.2,2.4). There is currently no explicit way to convey a curved spatial path. Subsequent frames or animation can also be used to convey movement.

4.6.4.2 Paths in animation

The term, path, has a history of use in computer animation. This use provided the inspiration for the notion of paths in REBUS, since at one time the goal was to animate scenarios. The particular work which was influential was Stasko's systems Tango and Polka [97, 98]. These systems provide programmers (especially beginning programmers) with a simple way to annotate their programs with procedure calls. This can support a clean separation of the algorithm and the algorithm animation code. With the focus of supporting animation, a set of predefined path types (e.g. fill, scale, move, color) are provided. In this approach, all paths are uniform in that they are a sequence of real valued pairs of numbers. The values only have meaning when associated with their path type. These typed paths are used as general means to change the appearance of a graphical object's depiction. This means that if we wish to support scenario animation, one thing we would need to provide is a means to map an object's attributes to the predefined path types of Stasko's paths.

4.6.4.3 Relationship to methods

In object-oriented design "methods" are associated with objects. In REBUS, paths are associated with the concrete orderings of an object's attribute values. Paths do not take the place of parameterized procedures like "methods", but if we consider some subset of the object's attributes as the parameters to "methods" we can use a path to describe the sequence of values passed to the method. That is, for an object, car, with an attribute, location, the abstract method might be "move(location)" while the simple path, "move", would be of type location and for example, have the path point, "garage."[4]

Parameterized procedures are the *lingua franca* of programmers. They are explicit and organize behavior from a programming/mathematical standpoint, but they are inappropriate for non-programmers and did not occur in the natural scenarios.

Without procedural notions from programming languages, there is some work that must be performed by the analysts to transform paths into methods. This can be done within the framework of a software engineering environment like ARIES [51]. Paths provide a basis for the traceability of methods. That is, an abstract method can be traced to one or more paths. Because paths encapsulate temporal information, one can even trace several procedure invocations to a single path.

[4]The car could also be modeled with another attribute, action. In this case a composite path would have both the "move" action and the "garage" location.

4.6.4.4 Path usage

Paths are used for describing object behavior in scenarios, so the ability to simply assign an object a path is needed. This is possible since the path types can be modeled as object attribute types. It is also reasonable to want to support paths organized in a hierarchy. For example, turns can be further divided into right turns and left turns.

Paths of length one are expected to be common since they are useful for re-using single states or events. Paths convey concrete sequences at a single level of abstraction over time. We do not assume there is any behavioral rationale or causal implication between *points* on the path and it is reasonable for the duration to be under-specified. It is not reasonable for a scenario writer to leave out a significant point in the path when expressing a scenario.

There are two ways to look at the attribute type, namely, continuous and discrete. For continuous types, ignoring details at the attribute level is necessary, but they still need to appear continuous in the animation. This can be handled with *trails* which are used in a manner closer to the use of paths in Stasko's [97] algorithm animation system. Spatial trails can be defined interactively and are useful for conveying smooth animation when the model is defined in terms of regions.

4.7 Behavioral elements

In REBUS behavioral elements are provided to model causal or conditional rules that are communicated in scenarios. There are two semantic formats for these rules: *Triggers* are used to convey the semantic notion of causal activation and *restrainers* convey the semantic notion of causal impedance. In REBUS, behavioral building blocks are classified as either triggers or restrainers. More specifically the term *stimulus* is used to define a building block classified as a trigger[5] and the terms *inhibitor* and *prohibitor* are building-blocks classified as restrainers.

[5]The rationale for two terms is that REBUS may evolve to have more than one form of trigger (e.g. explicit constructor, destructor, and messaging building-blocks).

Triggers and restrainers are used to express a conditional or constraining relationship between the current state[6] (with paths used for historical states) and the next state (with paths used for the future states). Note that paths add significant semantic expressivity when used in conjunction with triggers and restrainers.

Before defining the details of the stimulus, inhibitor and prohibitor building blocks, there are some common components called "sides." The term "side" reflects a partial spatial layout for the components. Thus "left-side" and "right-side" are the basic sides. The layout is partial because a prohibitor also has an "until-side." These sides are composed of concepts which have attributes (e.g. objects, regions, etc.). When located in the context of a trigger or restrainer, a concept has two other fields: a *count*, which reflects quantification and a *naming label* which can be used to distinguish multiple concepts of the same type.

The count field supports a variety of notions. For example, 'there exists' is reflected in a count of 1 or more and 'Does not exist' is a count of 0. Count can be built as logical formula. For example, There exists more-than four of an object is expressed with count greater than 4.

The notion of 'For all' is implicit. Anytime a left side is true about the concrete world model, the trigger or restrainer conditional has been satisfied for all the unique occurrences in the world model. This is because of the execution semantics which is described in section 4.7.4.1.

The concepts (e.g. objects, regions, etc.) in any of the 'sides' of a trigger or restrainer have an extra field for local naming. Also, the concept attributes have a 'significance' flag. Given a concept has many attributes, this flag is used for focus. Thus, only when the flag is set is an attribute's value (or path, for objects) considered important to the condition. For right sides there is an additional qualifier, namely, *comes from*. This qualifier is useful for stating that the value/path on the right side is the result of a copy or formula applied to some attributes' value from another side.

[6]State is meant to include system and environment state

In REBUS paths can be used in the side of triggers and restrainers to represent histories and futures. Left and until side conditions for objects can only hold at the end of all the attribute paths. Paths on a *right side* reflect future values.

4.7.1 Stimulus

A stimulus has a name, category, description, and two 'sides'. The left-side is the triggering situation for the right-side to occur. That is the *stimulus left side* is the triggering condition and the *stimulus right side* describes the behavior triggered. This is temporally formalized as:

If left-side is true at time t then do right-side at time $t + \Delta t$, $\Delta t > 0$.

Create and Destroy: In REBUS construction and destruction of domain concepts are specified as a stimulus. They are handled by setting the count value for the object on the left side. If the right side does not have an object on the left side, then the object on the right side is created when its count is set to 1 or more. To destroy an object the right side and left side contain the same object, but the right side has count zero.

This approach has the limitation that construction and destruction are not explicit building blocks of REBUS. The rationale for not making them building blocks comes from their absence in the concepts seen in the domain expert's scenarios. Software experts place more importance on such constructs. Automated support could be provided for the software experts to filter the stimuli for instances of construction and destruction.

4.7.2 Inhibitor

A inhibitor has a name, category, description, and two 'sides'. The left side is a restraining description for preventing the right side. That is the *inhibitor left side* is the inhibiting condition for the *inhibitor right side* which describes the behavior immediately restrained. This is temporally formalized as:

If left-side is true at time t then can't do the right-side at time t.

4.7.3 Prohibitor

A prohibitor has a name, category, description, and three 'sides'. The left side causes the restraint of the right side until the until-side occurs. That is the *prohibitor left side* is the prohibiting criteria for the *prohibitor right side* which describes the behavior restrained until the *prohibitor until side* is true. This is temporally formalized as:

If left-side is true at time t, until-side is true at time t' $(t' > t)$, and until-side is false at times p such that $t < p < t'$ then can't do the right-side at anytime t'' $(t < t'' < t')$.

Inhibitors and prohibitors convey different meanings. To illustrate this, compare statements 1 and 2.

1. If the gauge's pressure is greater than 50 psi prohibit the valve from opening (until the gauge's pressure is not greater than 50 psi).

2. While the gauge's pressure is greater than 50 psi prohibit the valve from opening until the operator checks it.

It would be awkward to state some forms of restraint without both inhibitor and prohibitor constructs. This is illustrated by stating the prohibitor as the inhibitor: If (the operator has not checked the valve since its gauge's pressure was last greater than 50 psi) prohibit the valve from opening.

4.7.4 Discussion

4.7.4.1 Execution semantics

The basic execution semantics of triggers and restrainers is that they should "fire" simultaneously when the condition described by the left side is met. Expert or production systems have the same basic execution semantics [12]. With this execution semantics, there is the problem of *conflict*. That is, what is the correct right side behavior when two or more triggers or restrainers have satisfied their left sides and their right sides would impose conflicting outcomes. To actually execute a set of rules a variety of solutions to this exist, including ordering, heuristics, and most specific left side. For REBUS, restrainers have precedence over triggers. For the purposes of requirements envisaging, the approach taken is identifying and discussing such conflicts with the domain experts. This can be done by discussing and capturing scenarios which illustrate the conflict.

4.7.4.2 Temporal specification

One issue unique to REBUS triggers and restrainers is temporal specification. That is, paths and duration specifications make triggers and restrainers temporally expressive.

There are two ways that duration specifications are associated with triggers and restrainers. Both are via paths. First, a duration specification is part of a path's associated with an object's attribute fields. These objects are in the 'sides' of a trigger or restrainer. Second, triggers and restrainers can be thought of as events or activities (by abstracting away from 'side' details) which have duration. When consideration of the duration of a trigger (or restrainer) is important the trigger (or restrainer) should be modeled (with the same name) as a path or path-point which has a explicit duration specification.

We note that in order to achieve actual execution of triggers and restrainers, one will have to decide a-priori a smallest unit of time (Δt) at which to model the execution. The obvious choice is to base time on the smallest unit provided by the system clock, but different domains have different requirements. It appears inappropriate to impose such a choice during requirements envisaging, especially since more than one time point can be represented by a path and we believe spatial units can also be used to delineate time.[7]

4.8 Scenarios

As defined in chapter 1, *scenarios* are partial descriptions of system and environment behavior. The organization of a scenario in REBUS takes its form from storyboards. Each scenario has a name, category, description, duration specification, and one of more *frames.* Each frame has a name, number, description, and *depiction area.* The depiction area contains the concepts, e.g., domain objects, spatial regions, and possibly further annotations.

The objects in the depiction area have an extra field for local naming and their attributes can have values or paths associated with them. With these paths, a frame can encapsulate more than a single state or time step.

Frames are related temporally, i.e. they are a temporal ordering.[8] Frames are also related causally, that is, triggers and restrainers occur between frames.[9]

4.9 Chapter summary

This chapter contains a detailed definition of a central contribution of this thesis: a domain-independent, semantically rich, representation for within-scenario concepts. It also contains a definition of a scenario, organized as a storyboard, which contains the within-scenario concepts. The representation was designed with concern for the target characteristics described in chapter 2. Its organization follows the conceptual

[7]In order to animate behavior, a smallest unit for the spatial coordinate system must be defined. This is a similar decision to that of smallest temporal elements. For raster displays, this amounts to the dimensions of the addressable pixels.

[8]There is symmetry between frames and simple paths. In viewing *between*-scenario notions, abstraction realizes this symmetry i.e. use the scenario name to correspond to a path name, and the frame names, to correspond to the paths values.

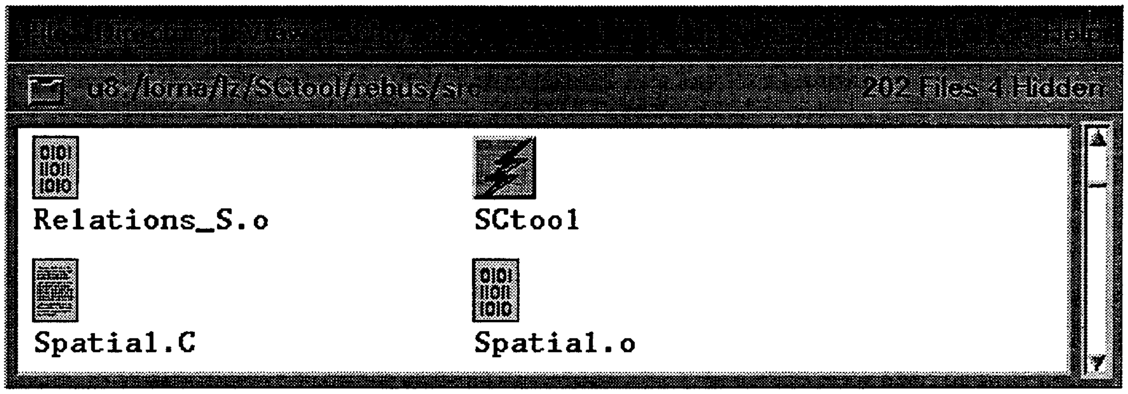
[9]It appears that there can also be triggers and restrainers in *Between*-scenarios relations.

framework of objects, units of measurement and types, spatial elements, temporal elements, and behavioral elements. These are further divided into building blocks.

To use the representation, one maps a concept from the application domain onto an appropriate building block. The coverage provided by this representation is also considered relative to concepts found in natural language and other relevant representations.

Chapter 5

An automated tool for scenarios



5.1 Introduction

This chapter describes a program, called SCtool, for capturing and manipulating scenarios. SCtool is a prototype which demonstrates the feasibility of providing scenario writers with suitable automated support. SCtool instantiates the REBUS representation as an automated tool with a graphical interface.

SCtool provides the scenario writer with a collection of editor and catalog dialogs. Each of the REBUS building blocks has its own editor (e.g. an object editor, a region editor, a simple path editor, a stimulus editor). To organize the domain concepts modeled with REBUS, catalogs are provided for each aspect of the conceptual framework (e.g. an object catalog, a spatial catalog, a triggers and restrainers catalog). There is also a scenario editor and a scenario catalog.

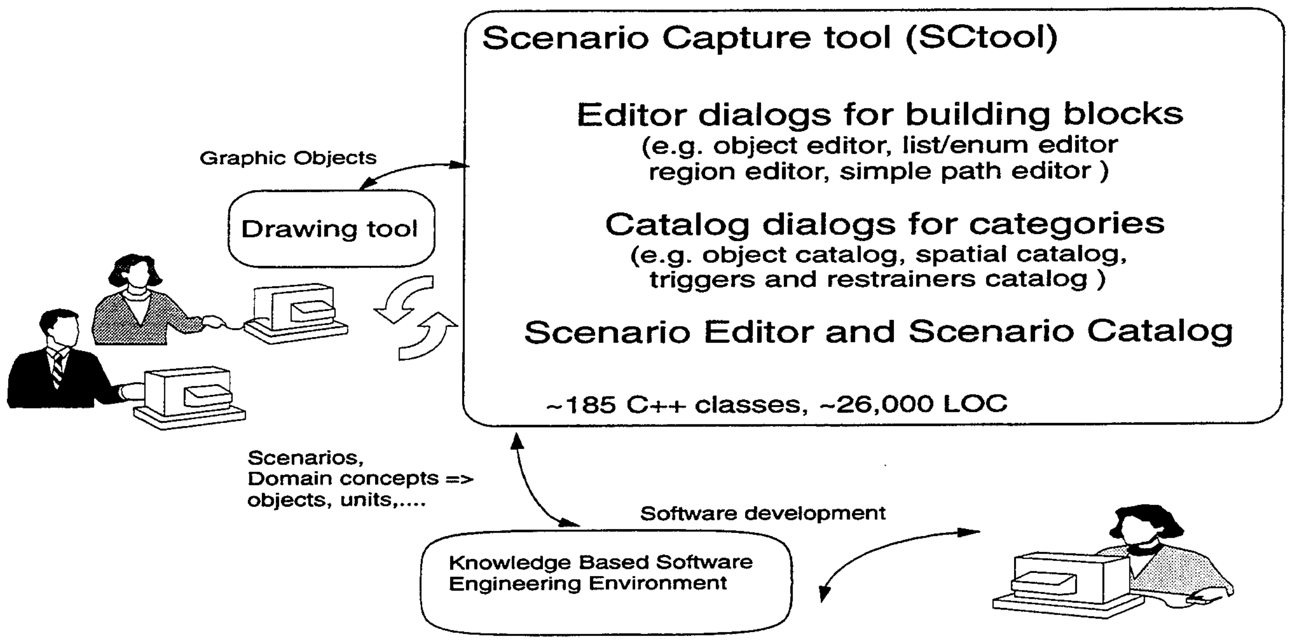

Scenario Capture tool (SCtool)
Editor dialogs for building blocks
(e.g. object editor, list/enum editor
region editor, simple path editor)
Catalog dialogs for categories
(e.g. object catalog, spatial catalog,
triggers and restrainers catalog)
Scenario Editor and Scenario Catalog
~185 C++ classes, ~26,000 LOC
Graphic Objects
Drawing tool
Scenarios,
Domain concepts =>
objects, units,....
Software development
Knowledge Based Software
Engineering Environment

Figure 5.1: SCtool overview

Since the goal is to support an iterative, opportunistic and ill-structured process, editors and catalogs are available to the scenario writer at any time (i.e. they are non-modal). The implementation currently provides one instance of each catalog, but multiple editors can be opened as needed to support the viewing and manipulation of domain concepts and scenarios.

An overview of SCtool is shown in Figure 5.1. The figure shows domain and software experts collaboratively working with SCtool and a drawing tool. It also illustrates the idea that SCtool could someday connect to knowledge based software development tools.

The next section contains implementation information. It is followed by descriptions of the various dialogs. This description includes details of the prototype's usage and "rough edges" as well as discussion of implementation decisions and rationale. Throughout this chapter, the figures illustrate SCtool with example domain knowledge from the telephony scenarios described in chapter 1.

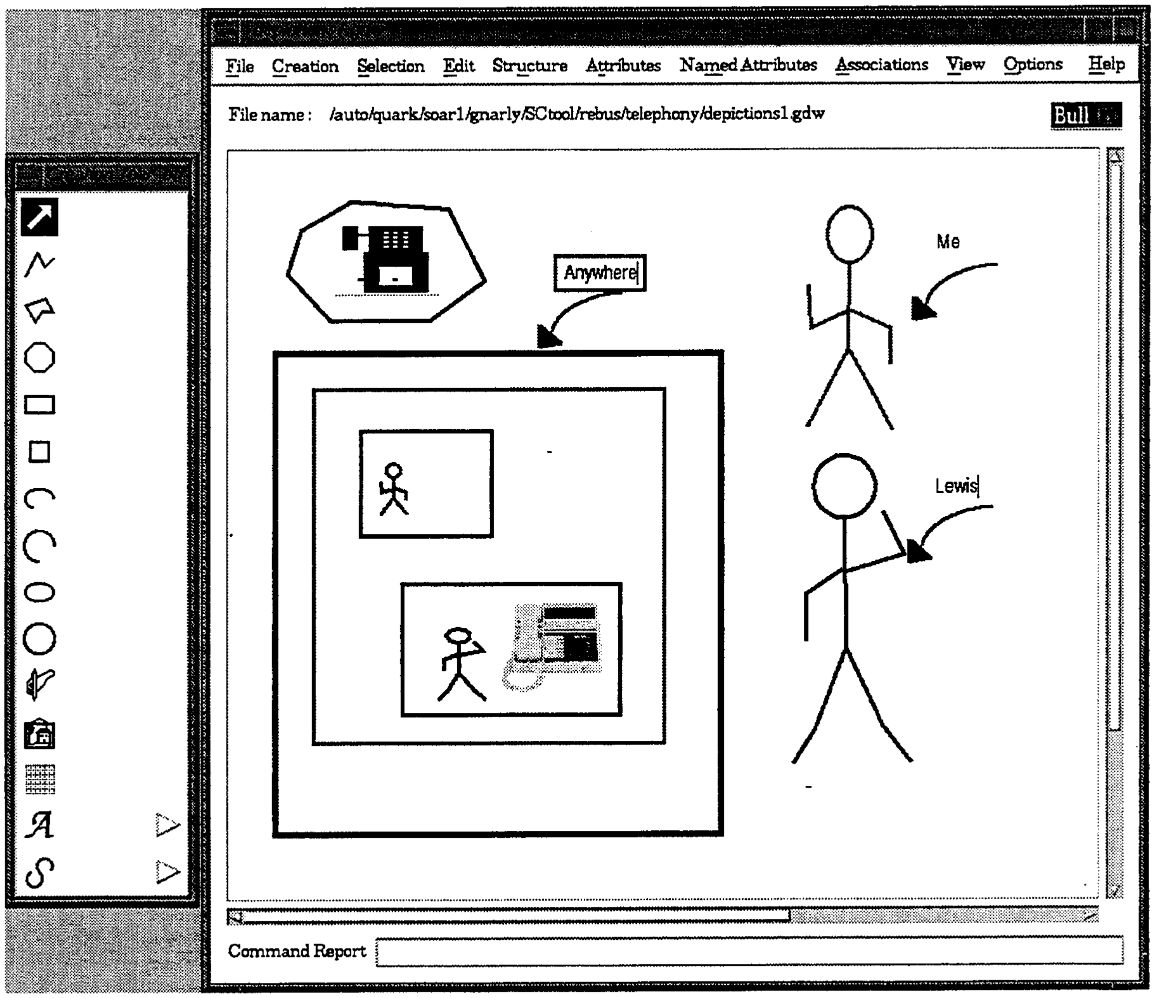


Figure 5.2: The GoDraw graphics editor

5.2 Implementation information

SCtool was implemented in the C++ programming language [100]. The following supporting libraries were used: The X Window System libraries [20], the OSF/Motif^{TM} Widget set [78], the Wcl Table Widget [96], and the GoPATH [21, 22] structured graphics libraries. SCtool has about 185 C++ classes and 26,000 lines of code (not including the libraries, but including code generated by a interface builder).

A part of GoPath, the GoDraw editor, was used as a stand alone graphical drawing tool (See figure 5.2). GoDraw was used as a basic drawing program with

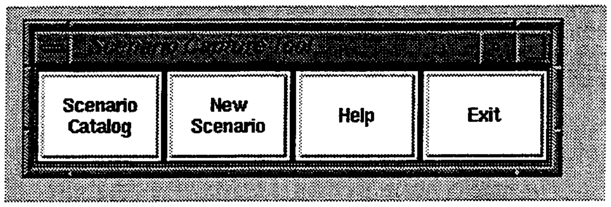


Figure 5.3: The initial SCtool dialog

which depictions could be created, manipulated and then imported and exported to SCtool. The GoPath library was also used for saving and restoring data to textual files in an object-oriented format.

5.2.1 Initial dialog

SCtool has an initial dialog which supports the basic functionality of accessing scenarios and exiting the program. As shown in Figure 5.3, SCtool also has a button for "Help." While "help" is a standard button in many of the dialogs, it is not an implemented feature of the prototype. Thus, the users in the evaluation which will be described in chapter 6 must ask for help or guess. Selecting "Scenario Catalog" from the initial dialog creates[1] the scenario catalog dialog.

5.2.2 Catalogs

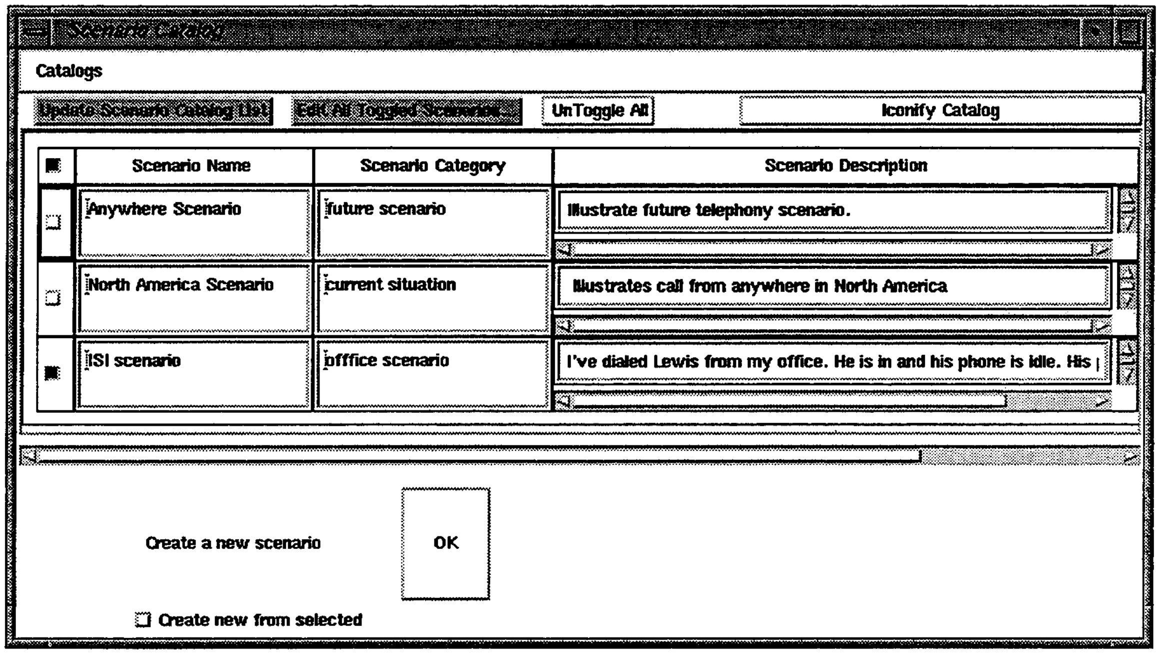
SCtool has a basic catalog dialog for scenarios as well as for objects, units of measurement/types, spatial elements, temporal elements, and behavioral elements. Minimally, each catalog provides functionality to browse and retrieve existing concepts.

To illustrate a catalog dialog, the "Scenario Catalog" appears in figure 5.4.[2] The figure shows a list of scenarios. The upper portion has a "Catalogs" menu bar item (which will be explained shortly). Below that there are four buttons. The first updates the catalog's list of entries.[3] The next two buttons are used in reference to

[1]If the catalog has already been created, the catalog is raised to the forefront.

[2]The telephony concepts presented in the figures are for illustrative purposes, so, for example, initially there are no scenarios in the scenario catalog.

[3]Currently, there is no database beneath SCtool and the catalog is not notified when new items have been created and saved.


Scenario Catalog
Catalogs
Update Scenario Catalog List
Edit All Toggled Scenarios...
UnToggle All
Iconify Catalog

	Scenario Name	Scenario Category	Scenario Description
☐	Anywhere Scenario	future scenario	Illustrate future telephony scenario.
☐	North America Scenario	current situation	Illustrates call from anywhere in North America
■	ISI scenario	offfice scenario	I've dialed Lewis from my office. He is in and his phone is idle. His

Create a new scenario
OK
☐ Create new from selected

Figure 5.4: The scenario catalog

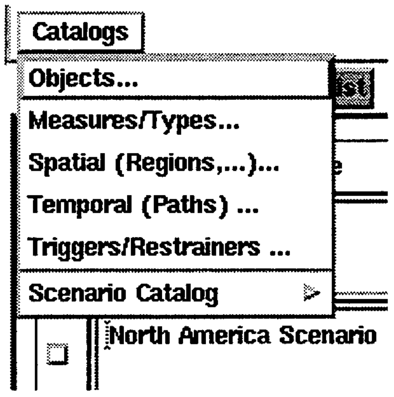


Figure 5.5: A catalog pulldown menu

the toggle buttons next to each scenario in the list. One is the command to open all the toggled scenario's. The other is a command to "un-toggle" all the toggled items. The last button iconifies the dialog. The lower portion of the catalog dialog has an area to selectively create concept editor dialogs. For example, the scenario catalog has a button to create a new scenario.

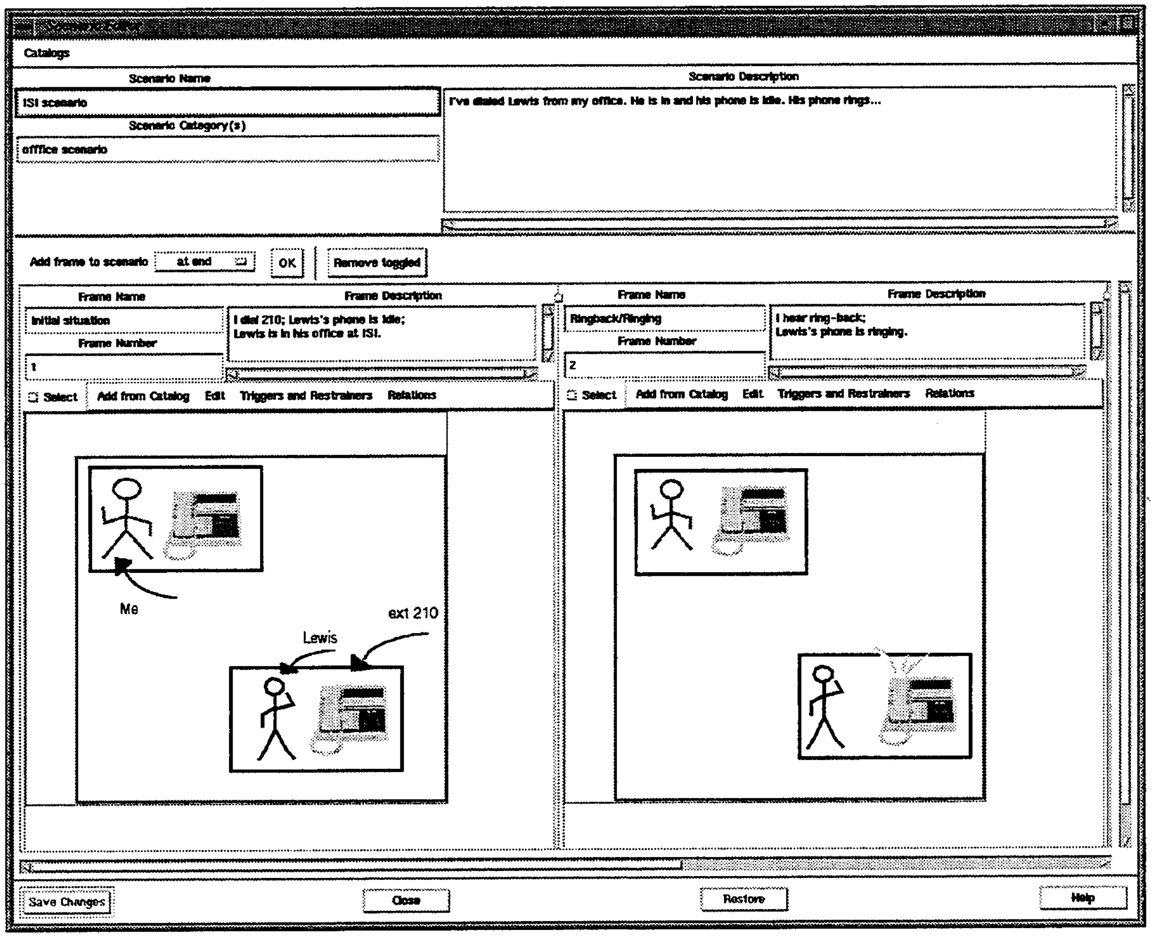
Throughout SCtool, there is a "Catalogs" menu bar item. This pulldown menu is shown in figure 5.5. This menu allows the user to create and then raise catalog dialogs to the forefront.[4]

Future versions of SCtool should provide more features in terms of catalog support. Currently, the list of catalog items is in the order the items were first saved, but other orderings or presentations could be provided. As the number of items in a catalog gets large, mechanisms to query and retrieve subsets will be needed. Since catalogs are also the means with which users create new concepts, facilities to create or modify concepts based on existing concepts should also be provided, thus allowing the user to easily create specializations and variants. We envision that the scenario catalog, in particular, will need to support the various *between*-scenario relations.

5.2.3 Scenario editor

The Scenario Editor is shown in figure 5.6 with data based on scenario 3 from chapter 1. The top part of the dialog allows the scenario writer to name, categorize, and textually describe the scenario. The middle section of the dialog is the frame editor. Frames can be added or removed as needed. In the current implementation, a newly created scenario starts without any frames. Two frames are visible in the figure. The lower section of the editor contains buttons for saving changes to the scenario, closing the dialog (with the option of saving when modified), as well as restoring the scenario from the last saved version.

[4]Technically, in the X window system, this duplicates functionality provided by a window manager.



Scenario Name: ISI scenario

Scenario Category(s): offfice scenario

Scenario Description: I've dialed Lewis from my office. He is in and his phone is idle. His phone rings...

Frame Name: Initial situation

Frame Number: 1

Frame Description: I dial 210; Lewis's phone is idle;
Lewis is in his office at ISI.

Frame Name: Ringback/Ringing

Frame Number: 2

Frame Description: I hear ring-back;
Lewis's phone is ringing.

Figure 5.6: A Scenario Editor Dialog

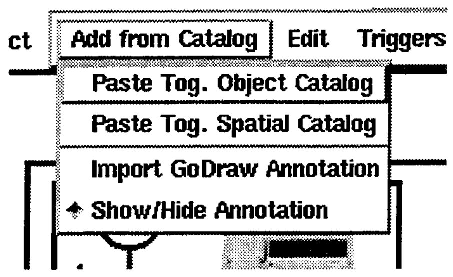


Figure 5.7: The "Add from Catalog" pulldown menu

5.2.3.1 Each frame

The top part of each frame allows the writer to enter a name, a number and a description. Frames contain the details of the concepts participating in the scenario as well as annotations.

To add items to the frame's depictive area one would ideally locate it from a catalog, drawing editor, or even another scenario then select, drag and drop a copy onto the frame. Currently, only cut and paste is supported for this task.

Each frame has a menu bar which contains four pulldown menus: Add from catalog, Edit, Triggers and Restrainers, and Relations. In figure 5.7, The Add from Catalog pulldown is shown. The writer has selected Lewis from the Object Catalog by toggle selecting the button in the left most column of the catalog and then pressing the button labeled "Paste Tog. Object Catalog." The writer can then interactively place the selected object (or objects) in the frame's depiction area. Default depictions (e.g. for objects two rounded boxes with the concept's name) are provided for the initial paste operation. See figure 5.9. The user can change the depiction via the object's popup menu.

The concepts in a frame's depiction area can be selected and manipulated. Figure 5.8 shows a popup menu associated with an object's depiction. The popup provides description information about a concept, e.g. its classification in REBUS, its name, its category, and its local name (if needed). It also provides access to a local values/path editor and a list of depictions. An example of the values/path editor is shown in figure 5.10. The attribute value/path editor provides a view of the object "Lewis." This editor allows the user to associate specific values or paths

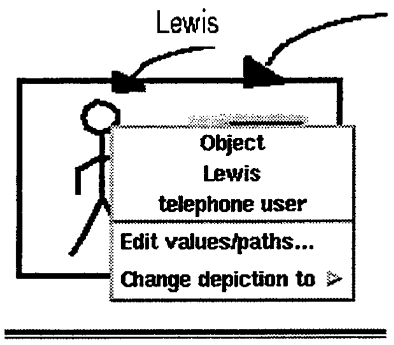


Figure 5.8: Selecting an object

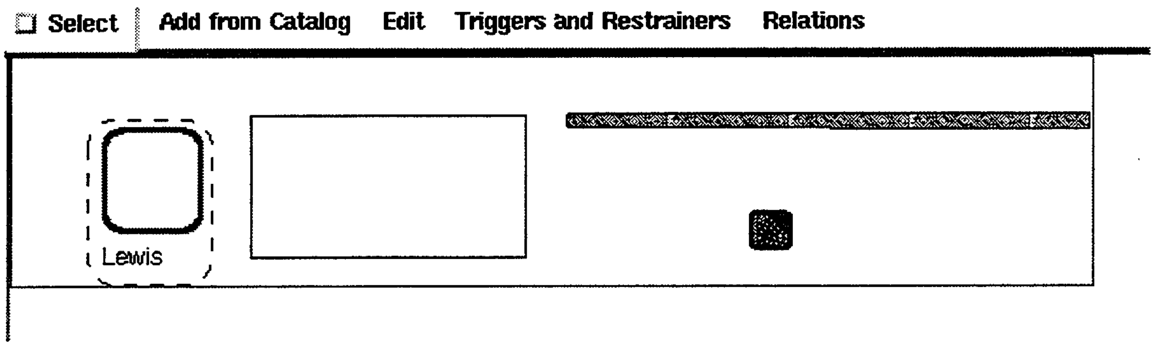
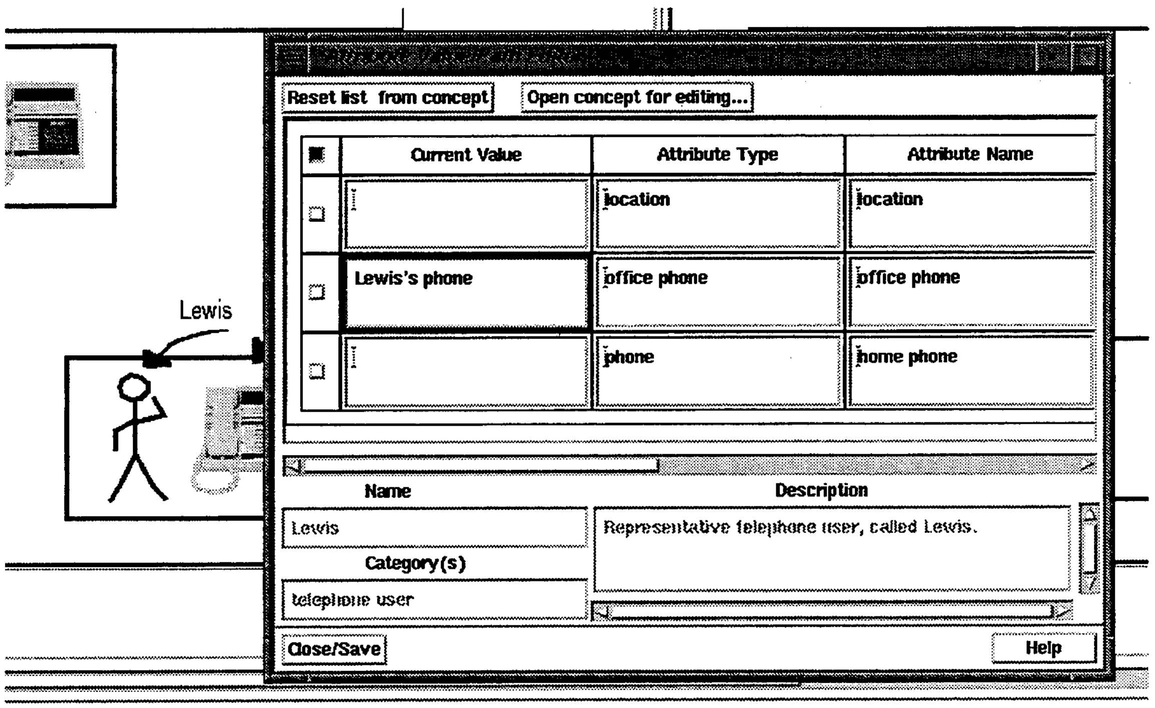


Figure 5.9: Default depictions - object, region, boundary, landmark



	Current Value	Attribute Type	Attribute Name
☐		location	location
☐	Lewis's phone	office phone	office phone
☐		phone	home phone

Name: Lewis

Category(s): telephone user

Description: Representative telephone user, called Lewis.

Figure 5.10: Editing the attribute values of an object

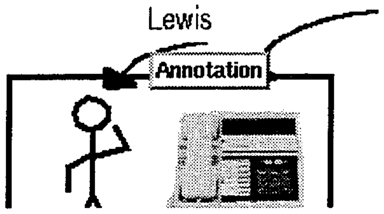


Figure 5.11: Selecting an annotation

to the attributes of an object. Paths are associated with an object by typing the path name (instead of an actual value) into the column labeled "Current Value." The figure illustrates that the object "Lewis's phone" is the current value of Lewis's office phone.

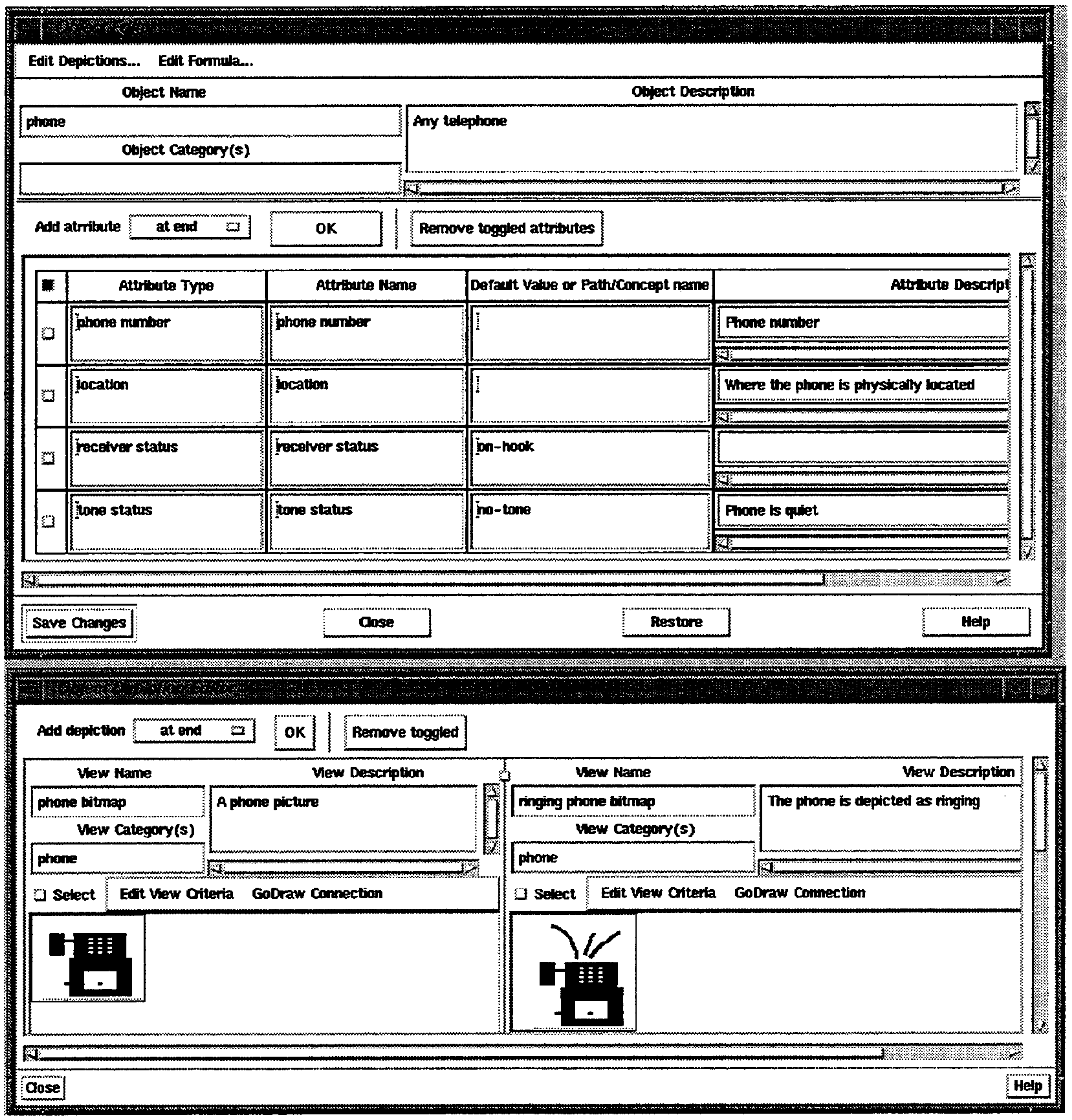
Annotation of the frame with structured text or graphics is supported with cut and paste operations between the GoDraw editor and the frame via the menubar item labeled "Import GoDraw Annotation." These annotations can be shown or hidden. Annotations are especially useful for visually highlighting parts of a concept's state.

5.2.4 Object editor

The Object Editor is shown in Figure 5.12. Attributes can be added or removed. Selecting the Edit Depictions button opens the Object Depiction Editor. The object depiction editor lets the writer add and remove views. The graphics in each view are created with GoDraw.

5.2.5 Measurement and types editors

The units of measurement and types dialogs are shown in figure 5.13 and in the subsequent figures. The unit type, list/enumeration, named quantity, conversion, and coordinate system editors are relatively straightforward implementations of REBUS. Each dialog has save and restore functionality. The list/enumeration and coordinate system dialogs support the addition and removal of list elements or axes, respectively.



Object Name: phone

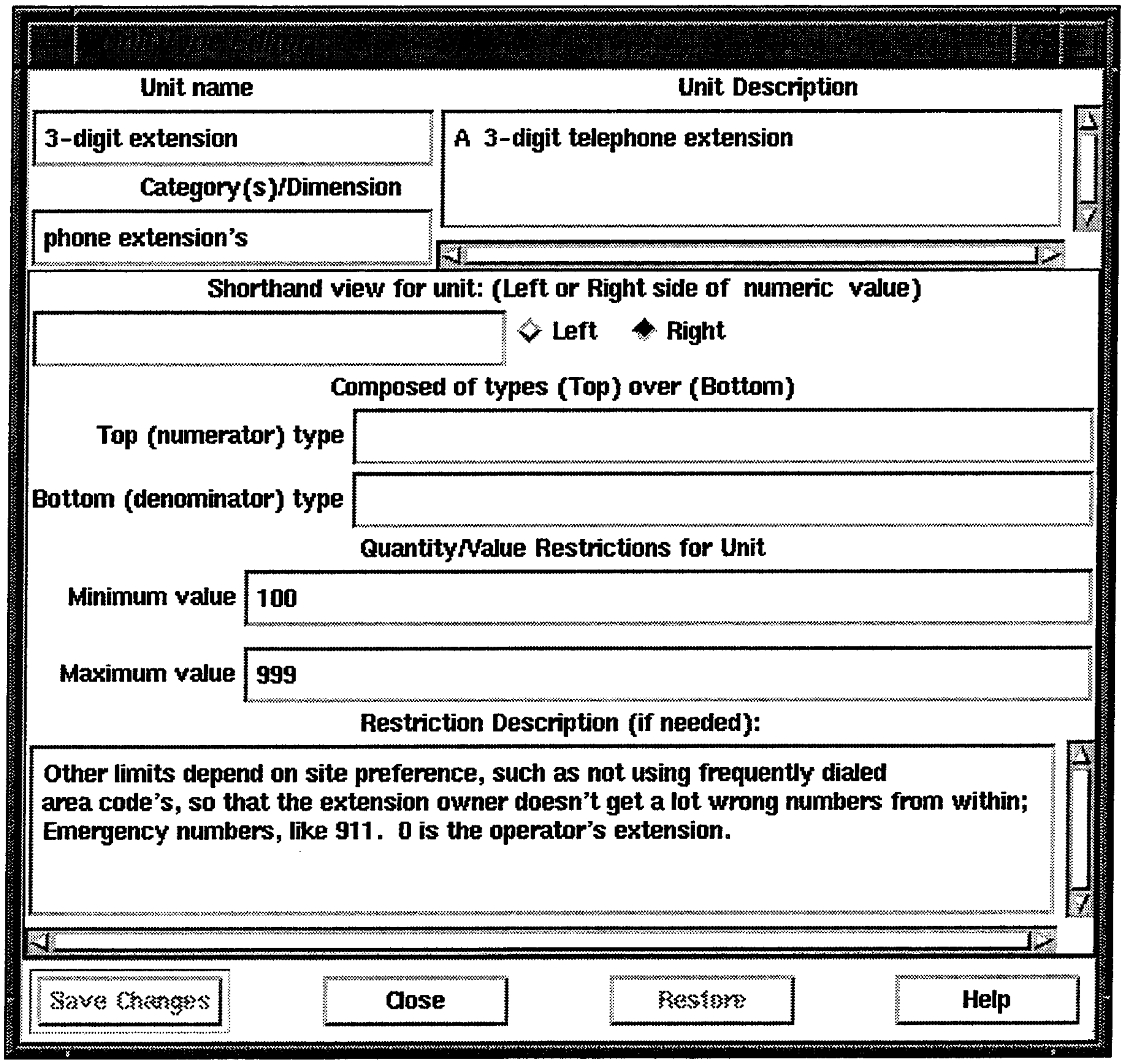
Object Category(s):

Object Description: Any telephone

	Attribute Type	Attribute Name	Default Value or Path/Concept name	Attribute Descript
☐	phone number	phone number		Phone number
☐	location	location		Where the phone is physically located
☐	receiver status	receiver status	on-hook	
☐	tone status	tone status	no-tone	Phone is quiet

View Name	View Description	View Category(s)
phone bitmap	A phone picture	phone
ringing phone bitmap	The phone is depicted as ringing	phone

Figure 5.12: Object Editor and Object Depiction Editor



Unit name: 3-digit extension

Unit Description: A 3-digit telephone extension

Category(s)/Dimension: phone extension's

Shorthand view for unit: (Left or Right side of numeric value)

Left [✓ Right]

Composed of types (Top) over (Bottom)

Top (numerator) type:

Bottom (denominator) type:

Quantity/Value Restrictions for Unit

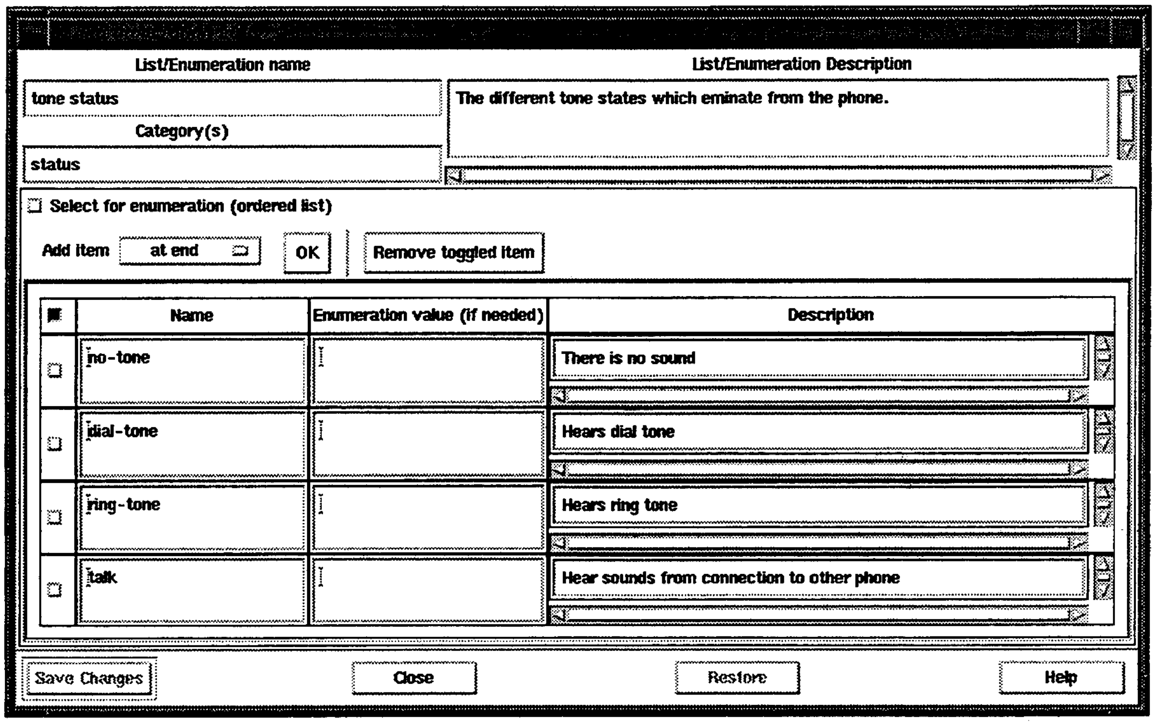
Minimum value: 100

Maximum value: 999

Restriction Description (if needed):

Other limits depend on site preference, such as not using frequently dialed
area code's, so that the extension owner doesn't get a lot wrong numbers from within;
Emergency numbers, like 911. 0 is the operator's extension.

Figure 5.13: Unit type editor



List/Enumeration name: tone status

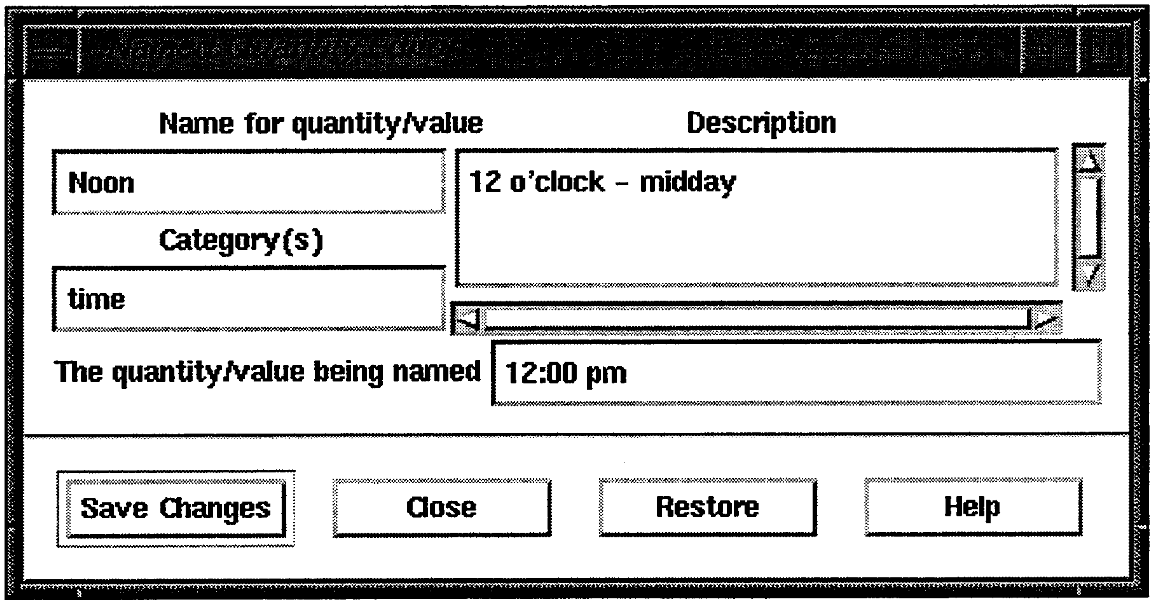
Category(s): status

List/Enumeration Description: The different tone states which eminate from the phone.

Select for enumeration (ordered list)

Name	Enumeration value (if needed)	Description
no-tone		There is no sound
dial-tone		Hears dial tone
ring-tone		Hears ring tone
talk		Hear sounds from connection to other phone

Figure 5.14: List type editor



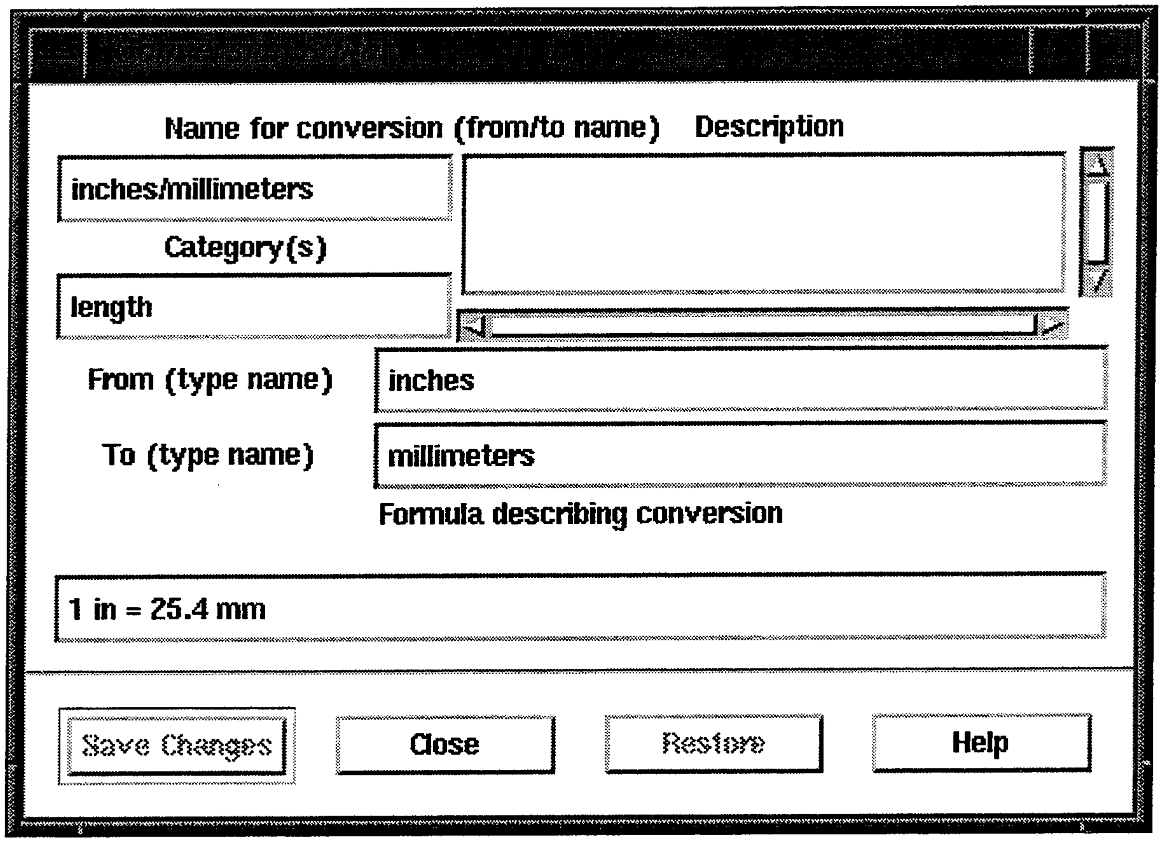
Name for quantity/value: Noon

Category(s): time

Description: 12 o'clock – midday

The quantity/value being named: 12:00 pm

Figure 5.15: Named quantity editor



Name for conversion (from/to name): inches/millimeters

Category(s): length

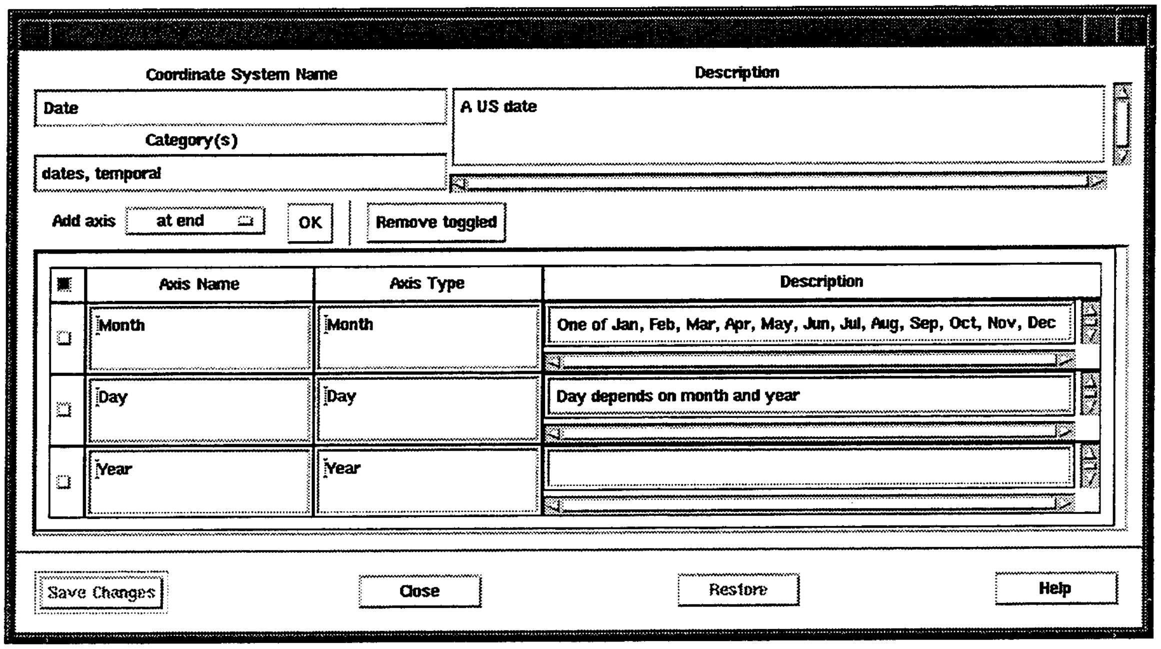
Description:

From (type name): inches

To (type name): millimeters

Formula describing conversion: 1 in = 25.4 mm

Figure 5.16: Conversion editor



Coordinate System Name: Date

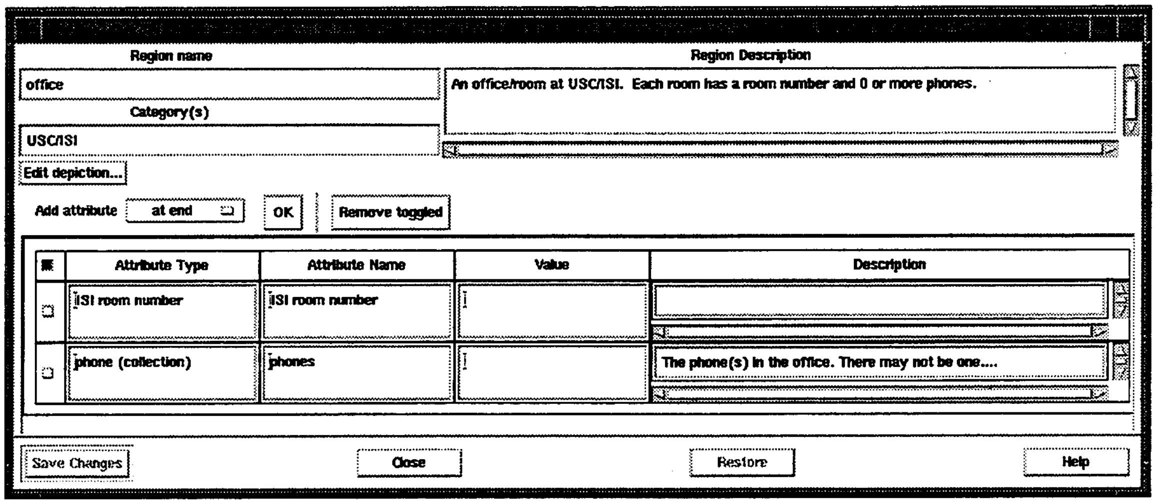
Category(s): dates, temporal

Description: A US date

Add axis: at end

	Axis Name	Axis Type	Description
	Month	Month	One of Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
	Day	Day	Day depends on month and year
	Year	Year	

Figure 5.17: Coordinate system editor


Region name: office

Category(s): USC/ISI

Region Description: An office/room at USC/ISI. Each room has a room number and 0 or more phones.

Add attribute: at end

Attribute Type	Attribute Name	Value	Description
ISI room number	ISI room number		
phone (collection)	phones		The phone(s) in the office. There may not be one....

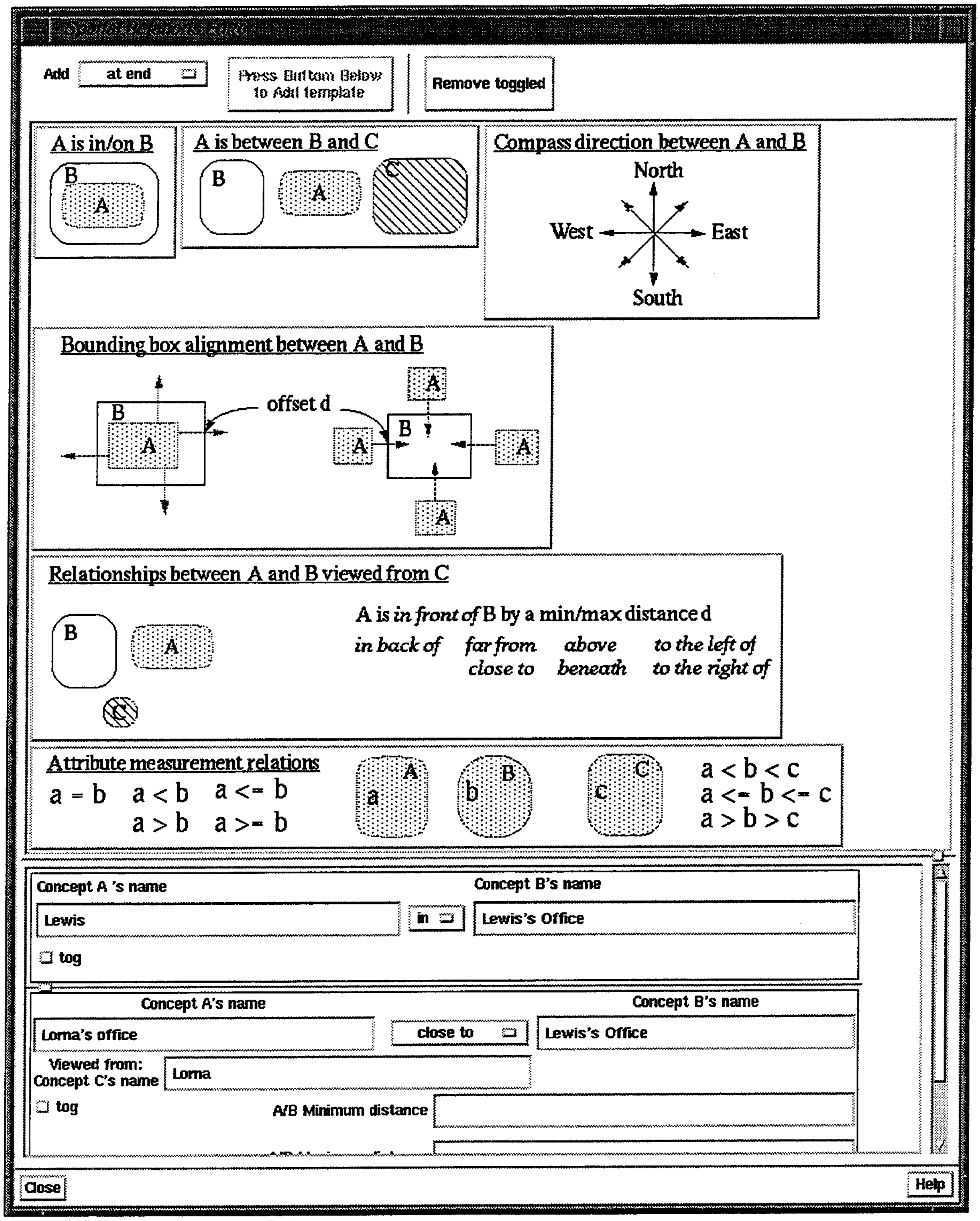
Figure 5.18: Region editor

5.2.6 Spatial concepts editors

Figure 5.18 shows the region editor. The region editor supports the naming and description of a region and the addition and deletion of attributes. Selecting the edit depiction button on the dialog brings up a spatial depiction editor. Spatial concepts have a single depiction.[5] Boundary and landmark editors are similar.

The spatial composite editor was not completed in the prototype, but it provides a dialog of spatial relations. The spatial relations dialog is shown in Figure 5.19. This dialog is currently accessible when editing a scenario's frame. The user of this dialog selects a spatial relation and then fills in the details of concepts for which the relation holds. Each scenario frame's spatial relations editor contains statements about the visual relationships that are important to the scenario writer in the frame. So in a frame, there may be visible spatial relations that are not considered important to the scenario writer.

[5]The rationale for only a single depiction is that spatial concepts do not have behavior, so their depiction shouldn't change.



Concept A 's name: Lewis
in
Concept B's name: Lewis's Office

Concept A's name: Lorna's office
close to
Concept B's name: Lewis's Office
Viewed from: Concept C's name: Lorna
A/B Minimum distance:

Figure 5.19: Spatial relations editor

5.2.7 Path Editors

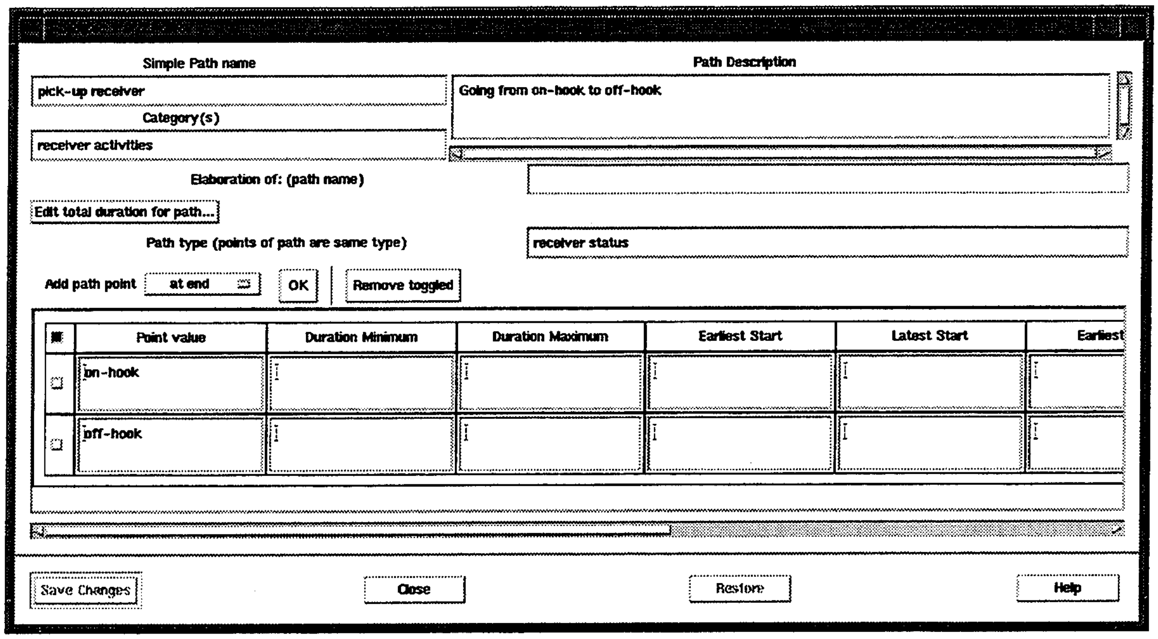
The simple path editor and composite path editor (with its associated temporal relations editor) are shown in figures 5.20 and 5.21, respectively. The simple path editor allows the user to add and remove path points. Path points are arranged in a list which is temporally ordered from top to bottom. In the figure, the path "pick-up receiver" is defined by the temporal ordering of points with the values "on-hook" and "off-hook". A button is provided to access a dialog for editing the total path duration specification. Each path point also has a duration specification. The field of the duration specification are located in the rows to the right of the point value column. Currently there is no support for checking the consistency of a path.

The composite path editor allows the user to add and remove paths from a list. This list's temporal ordering is defined by the temporal relations defined in the temporal relations editor. To use this editor the user selects a temporal relation and a template is provided to fill in the details of the corresponding paths. The example composite path in figure 5.21 joins two simple paths with the relation that one finishes before the other starts, thus picking up the receiver precedes hanging it up. This composite path defines the more abstract "Minimal call" temporal ordering of simple paths. The relation "A finishes before B starts" is specified by editing the dialog as shown in figure 5.22. This dialog is an editor for the temporal relations between the elements of a composite path.

The temporal dialogs illustrate the temporal concepts that can be expressed with REBUS, but further user interface work can be done. Currently, complex paths (ones with longer sequences of values or many relations) are difficult to visualize and manipulate. Alternative visual organizations such as timelines or graphs may facilitate the readability of complex paths.

5.2.8 Triggers and restrainers editors

The implemented triggers and restrainers dialogs are the stimulus, inhibitor, and prohibitor editors. The stimulus editor is shown in figure 5.23 and the inhibitor and prohibitor editors are shown in figures 5.24 and 5.25, respectively. The top part of



Simple Path name: pick-up receiver

Category(s): receiver activities

Path Description: Going from on-hook to off-hook

Elaboration of: (path name)

Path type (points of path are same type): receiver status

	Point value	Duration Minimum	Duration Maximum	Earliest Start	Latest Start	Earliest
	on-hook					
	off-hook					

Figure 5.20: Simple path editor

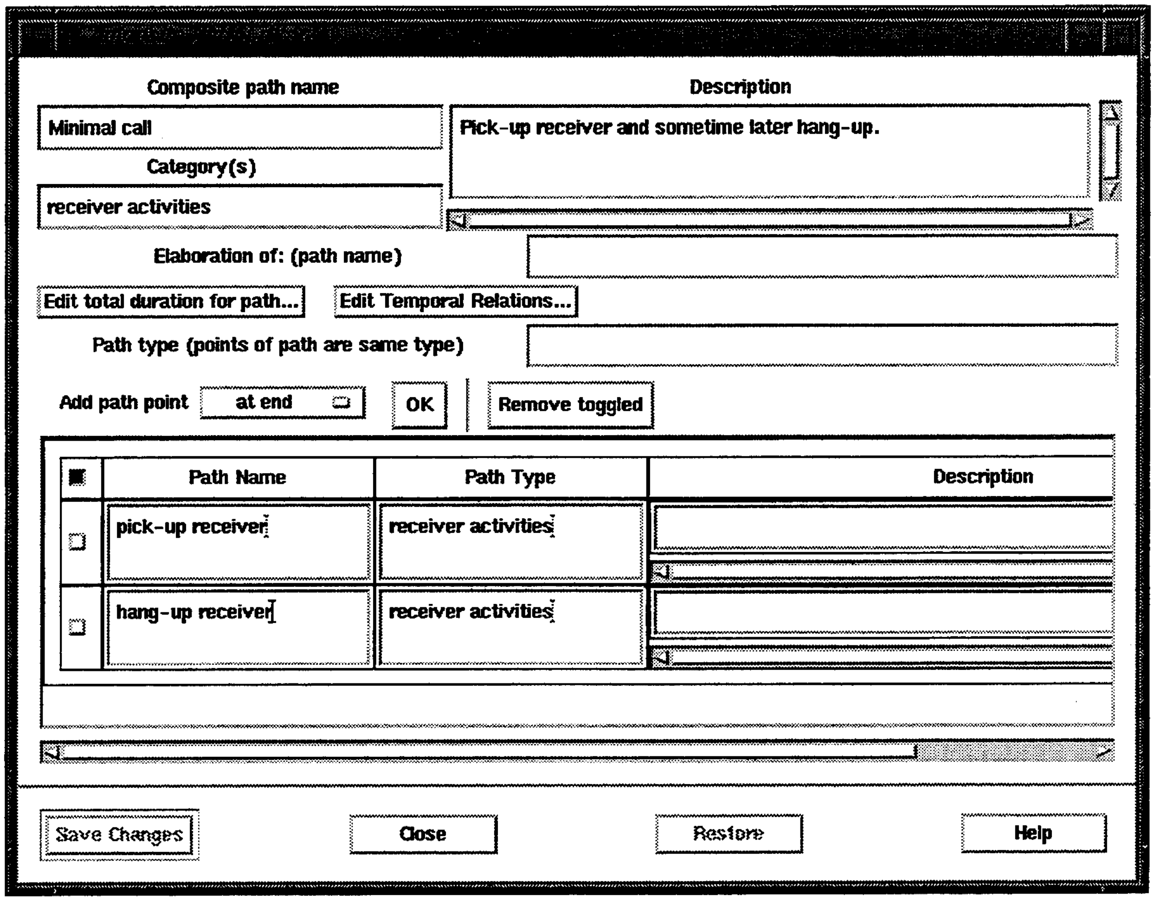
each dialog has the name, category and description fields and the majority of the dialog is used to edit the left, right, or until "sides."

The sides section is read from left to right. Due to limited screen space, this part of the dialog is implemented as a paned window with scrolled areas for the components of each "side." Currently, each side has limited functionality. The user must add concepts by selecting the location in the side and clicking on OK. This adds a template which the user then fills in with the concept's name, count, attributes, etc. Thus, the dialog is far from ideal.

Further work to support the selection of concepts from other locations (such as scenarios or catalogs) and place and edit them with the appropriate side specific information will be needed. In addition, graphical depiction can be used as a visual filter when textual details are consuming too much screen space.

5.3 Chapter summary

This chapter shows the prototype implementation of an automated tool for scenario capture (SCtool) based on REBUS. This prototype is intended to demonstrate the



Composite path name: Minimal call

Category(s): receiver activities

Description: Pick-up receiver and sometime later hang-up.

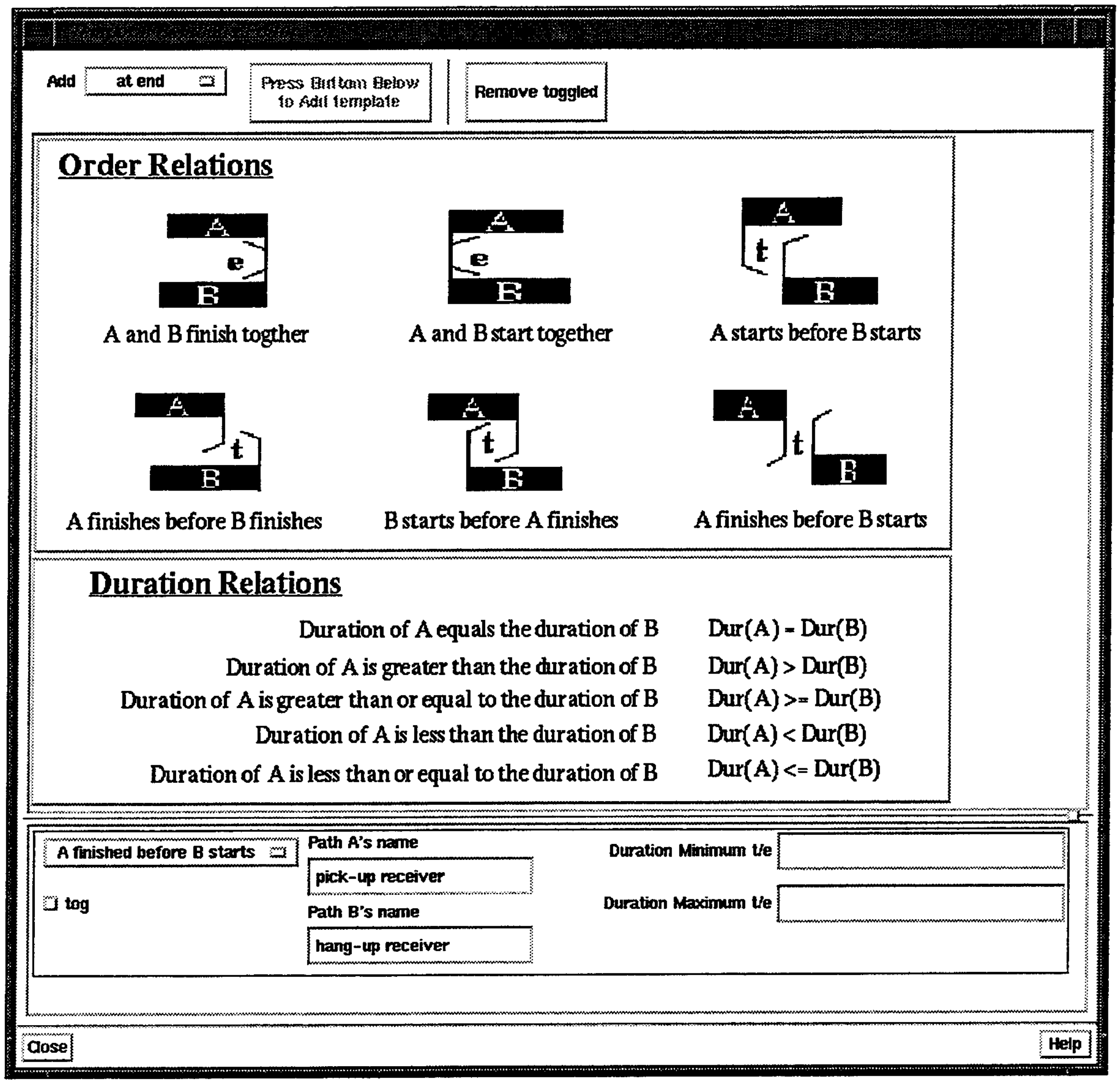
Elaboration of: (path name)

Path type (points of path are same type)

Add path point: at end

	Path Name	Path Type	Description
☐	pick-up receiver	receiver activities	
☐	hang-up receiver	receiver activities	

Figure 5.21: Composite path editor



Order Relations

A and B finish togther

A and B start together

A starts before B starts

A finishes before B finishes

B starts before A finishes

A finishes before B starts

Duration Relations

Duration of A equals the duration of B	Dur(A) = Dur(B)
Duration of A is greater than the duration of B	Dur(A) > Dur(B)
Duration of A is greater than or equal to the duration of B	Dur(A) >= Dur(B)
Duration of A is less than the duration of B	Dur(A) < Dur(B)
Duration of A is less than or equal to the duration of B	Dur(A) <= Dur(B)

A finished before B starts

❏ tog

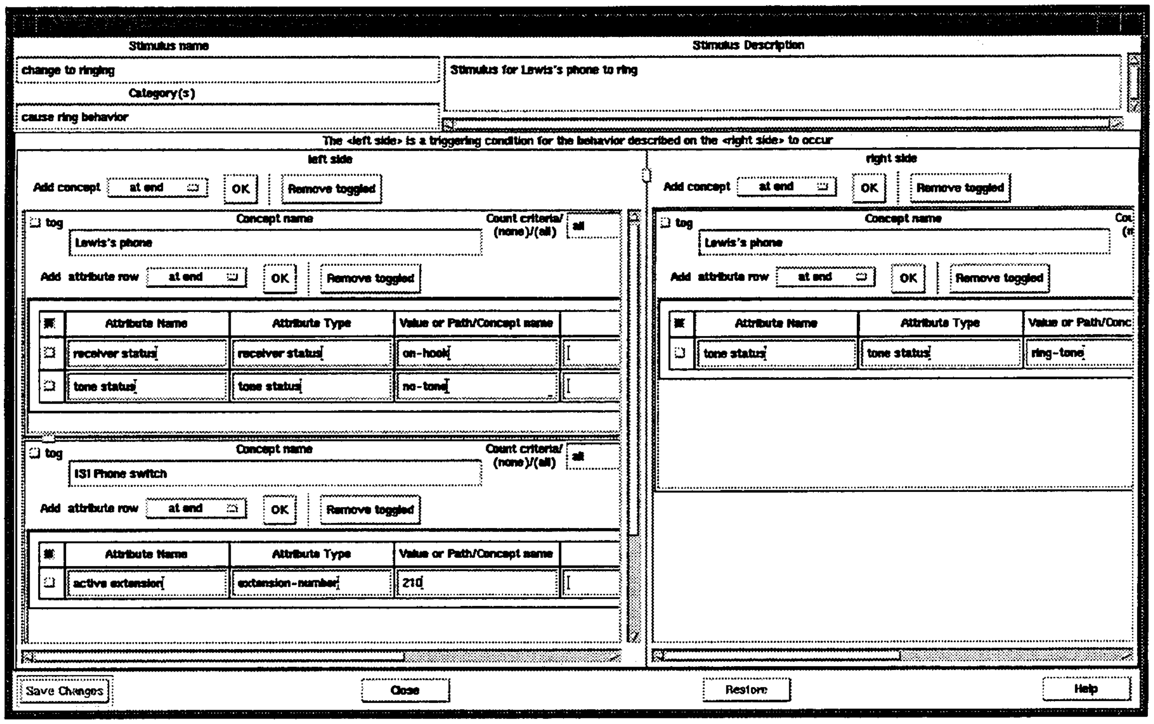
Path A's name: pick-up receiver

Path B's name: hang-up receiver

Duration Minimum t/e:

Duration Maximum t/e:

Figure 5.22: Temporal relations editor



Stimulus name: change to ringing

Category(s): cause ring behavior

Stimulus Description: Stimulus for Lewis's phone to ring

The <left side> is a triggering condition for the behavior described on the <right side> to occur

left side

Add concept: at end

Concept name: Lewis's phone — Count criteria/(none)/(all): all

Attribute Name	Attribute Type	Value or Path/Concept name
receiver status	receiver status	on-hook
tone status	tone status	no-tone

Concept name: ISI Phone switch — Count criteria/(none)/(all): all

Attribute Name	Attribute Type	Value or Path/Concept name
active extension	extension-number	210

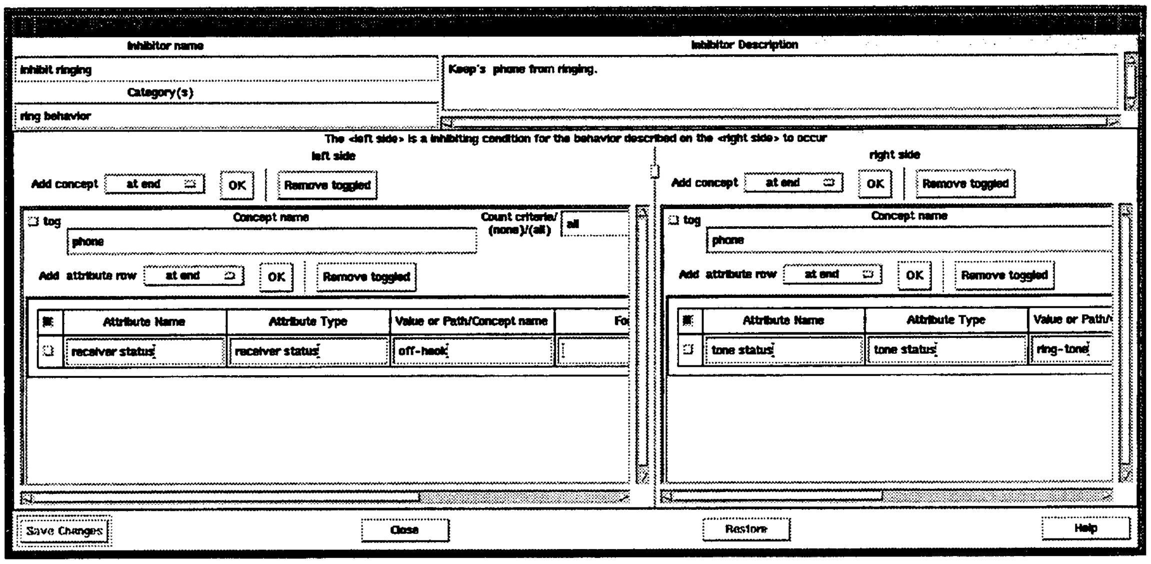
right side

Add concept: at end

Concept name: Lewis's phone

Attribute Name	Attribute Type	Value or Path/Conc
tone status	tone status	ring-tone

Figure 5.23: Stimulus editor



Inhibitor name: inhibit ringing

Category(s): ring behavior

Inhibitor Description: Keep's phone from ringing.

The <left side> is a inhibiting condition for the behavior described on the <right side> to occur

left side

Add concept: at end

Concept name: phone — Count criteria/(none)/(all): all

Attribute Name	Attribute Type	Value or Path/Concept name
receiver status	receiver status	off-hook

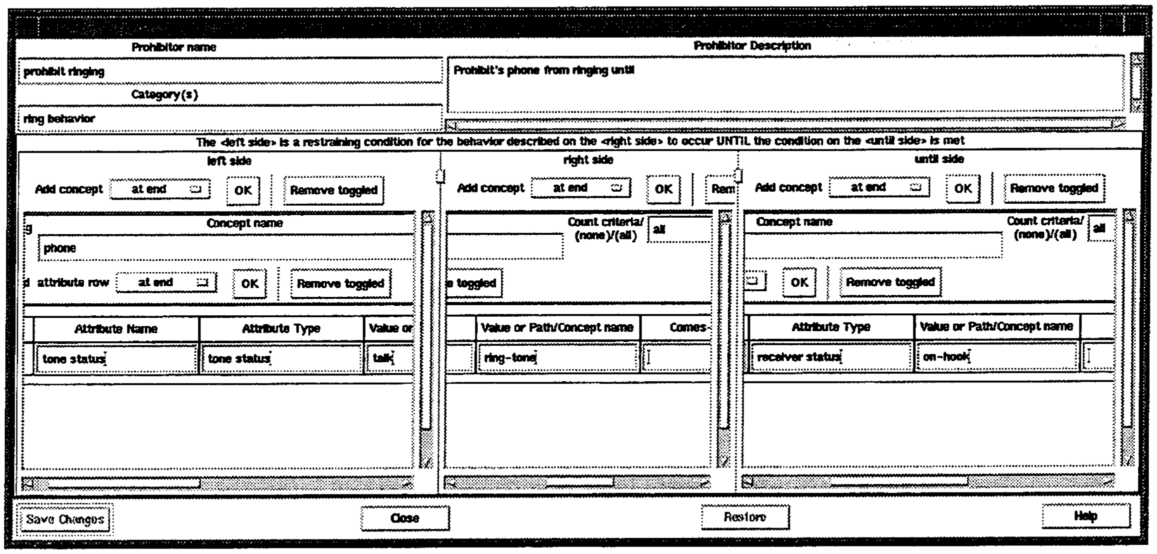
right side

Add concept: at end

Concept name: phone

Attribute Name	Attribute Type	Value or Path/
tone status	tone status	ring-tone

Figure 5.24: Inhibitor editor



Attribute Name	Attribute Type	Value or
tone status	tone status	talk

Value or Path/Concept name	Comes-
ring-tone	

Attribute Type	Value or Path/Concept name	
receiver status	on-hook	

Figure 5.25: Prohibitor editor

feasibility of providing a scenario capture tool which has a forms-based interface, has strong semantics, and is domain independent, but still structured around domain knowledge. The SCtool graphical user interface is presented and described.

Chapter 6

Evaluation

Colin Potts [81] suggests that researchers who wish to be taken seriously by practitioners need to adopt an "industry as laboratory" research methodology.

6.1 Introduction

It is important to get feedback on the use of REBUS and its prototype implementation, SCtool, in real world situations. While an evaluation of REBUS could utilize contrived tasks or controlled situations, valuable things can be learned from a study emphasizing external validity. SCtool was used by people carrying out their own tasks. This chapter describes the use of REBUS and SCtool in a real world application domain driven by the needs of a real world knowledge acquisition problem.

This application domain is the operational control of NASA's Deep Space Network (DSN). This domain was not among the domains studied while developing REBUS, but since it fits the overall characterization of a domain in which objects interact with their environment, it is an appropriate setting in which to use REBUS.

The evaluation presented here is formative in that we are trying to obtain information about the design of the prototype, rather than to measure outcomes such as improvements in resulting requirements documents. REBUS and SCtool's strengths and weaknesses are evaluated in a real knowledge acquisition meeting, by having someone other than myself use REBUS and SCtool, and by using REBUS and SCtool in a different context of acquiring scenarios when compared to the intelligent forces observational study.

There are several differences in context. In the case of intelligent forces, agents were to be developed from scratch and integrated with a graphics simulator also under development. For the DSN, operational systems exist and knowledge of the existing systems is gathered from various sources for the development of domain and task specific simulation capabilities. Also in the videotaped meetings of pilots and software experts, some of the software experts were at the early stages of understanding the domain, while the software experts in DSN have more domain experience and the domain experts have more programming experience.

First, this chapter will briefly describe the DSN application domain. Then further details of the requirements acquisition situation, i.e. the participants'[1] background and training, the preparation prior to the meeting, and the background context and scenarios which needed to be collected during the meeting in which SCtool was used. This meeting was videotaped to capture the communication between the domain and software experts. This chapter also documents and summarizes the data which were collected with SCtool. Finally, we present details and analysis of the evaluation experience.

6.2 The application domain

The following description comes from Hill et al.[43]:

> The Deep Space Network (DSN) is a worldwide network of deep space tracking and communications complexes located in Madrid, Spain, Canberra, Australia, and Goldstone, California. Each of these complexes is capable of performing multiple missions simultaneously, each of which involves operating a communications link. A DSN communications link is

[1]a.k.a the subjects

a collection of devices used to track and communicate with an unpiloted spacecraft.

Currently, most of the tasks requiring the control of a DSN communications link are performed by human operators on a system called the LMC (Link Monitor and Control) system. The Operators are given tasks that involve configuring and calibrating the communications equipment, and then they monitor and control these devices while tracking a spacecraft or celestial body. The Operators follow written procedures to perform their mission tasks. A procedure specifies a sequence of actions to execute, where the actions are usually commands that must be entered via the link's monitor-and-control system keyboard.

Once issued, a command is forwarded to another subsystem, which may accept or reject it depending on the state of the subsystem at the time that the command is received. The Operator receives a message back from the subsystem indicating whether the command was accepted or rejected, and in cases where there is no response, a message saying that the command "timed out" is sent. These messages do not indicate whether the action was successful or what the results of the action were. Rather, the Operator has to monitor subsystem displays for indications that the action completed successfully and that it had its intended effects. It is common for commands to be rejected or for commands to fail due to a number of real-world contingencies that arise in the execution of a block or procedure.

For further project context, from 1991-1993 JPL developed a prototype, the "Link Monitor and Control Operator Assistant (LMCOA)" to improve Operator productivity by automating some of the functions of the LMC.

Hill et al. state that the LMCOA performs tasks by: (1) selecting a set of blocks which contain commands to execute, (2) checking whether a block's preconditions have been satisfied, (3) issuing the commands, and (4) subsequently verifying that the commands had their intended effects. The Operator interacts with the LMCOA by watching the blocks as they are being executed and pausing or skipping portions of the block that need to be modified for some reason. When a block fails, the LMCOA lacks the ability to recover on its own. Instead, the Operator is left to figure out how to recover from the failure.

6.3 JPL's need for REBUS/SCtool

Under development at JPL is a new version of the LMCOA that will include a component, REACT-P. This component will reactively generate new plans in response to failures or changes in goals initiated by the Operator.

JPL's knowledge acquisition problem is in gathering and validating the domain knowledge. The domain knowledge needed by REACT-P includes the following: (1) the blocks and their actions, (2) the preconditions for each action, (3) the effects or postconditions for each action, (4) the goals of each set of blocks, (5) a partial order among blocks, and (6) the dependencies among blocks. This knowledge is normally only found in an expert's skill base and not recorded in a declarative form. Fragments of this information exist in the procedure manuals and operating guides for the various devices, but much of it is undocumented and can only be deduced from experience interacting with the devices.

JPL saw REBUS/SCtool to be of potential benefit toward alleviating the knowledge acquisition problems encountered in developing the knowledge base for REACT-P. They surveyed various knowledge acquisition tools and representations. The existing tools and representations did not meet their needs so they chose to participate in this evaluation of REBUS/SCtool.

An overview of how JPL viewed REBUS/SCtool can be seen in figure 6.1 taken from [43]. In the figure, REBUS is shown to connect to two boxes, labeled "RIDES" and "TDN". These components were designed for knowledge acquisition prior to JPL's introduction to REBUS/SCtool. SCtool was seen as an additional tool to be placed in front of the others since it could be used to express the knowledge needed by both components. SCtool was not to take the place of the other components. Many

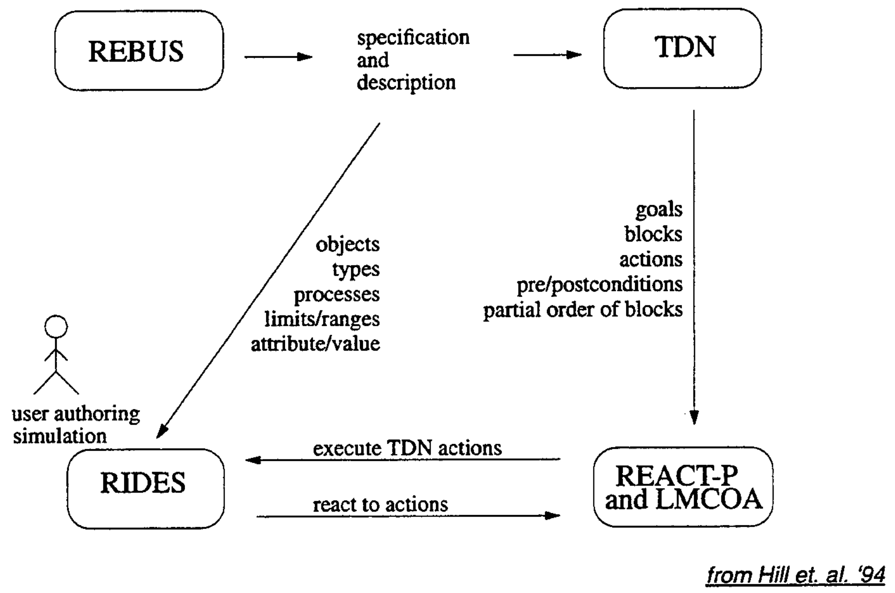


Figure 6.1: REBUS in the context of JPL's tools

of the features of these other tools are used to express programming level details (e.g. the screen location for an object is expressed as a pair of integers.). The collection of components (RIDES/TDN/REACT-P) are analogous to the possible connection that SCtool would have with a knowledge-based software engineering environment.

For example, the box labeled "RIDES" is the RIDES simulation authoring tool [65]. Hill et al. state that the RIDES graphical simulation authoring toolkit is being used to develop working models of the devices. JPL envisions the domain-specific simulations to serve two purposes: (1) to communicate with the subsystem engineers about how their system works, and (2) to test experimental prototypes for the DSN. RIDES is used to recreate the graphical user-interface to the LMC system and to model the devices with objects and attribute-values (represented at the detailed programming level, i.e. integers and characters). The simulations built in RIDES will be connected to REACT-P to provide a domain specific interface with which to evaluate the behavior of REACT-P. The testing of experimental prototypes is necessary, since access time to the DSN will be quite costly and extreme safety measures must be observed.

The second box labeled "TDN" is the Temporal Dependency Network (TDN) authoring tool. The TDN as a representation plays a significant role in the contexts in which REBUS/SCtool were used. Thus, TDNs will next be described in further detail.

6.4 The temporal dependency network (TDN)

The following details are from [26, 43].

> For encoding knowledge of the operator's tasks, JPL has developed a representation called a Temporal Dependency Network. It is used by the system engineers in order to express the basic block structure and control flow of the system directives used in operating the DSN LMC interface.

A TDN is a directed graph that incorporates temporal and behavioral knowledge and also provides optional and condition paths through the network. The directed graph represents the steps required to perform an operation. Precedence relations (step A has to happen before step B) are specified by the nodes and arcs of the network. The behavioral knowledge identifies system-state dependencies in the form of pre- and post- conditions. Temporal knowledge consists of both absolute (e.g. Acquire the spacecraft at time 02:30:45) and relative (e.g. Perform step Y 5 minutes after step X) temporal constraints. Conditional branches in the network are those performed only under certain conditions. These are the IF (this condition) THEN (do/don't do that action). Optional paths are those which are not essential to the operation, but may, for example, provide a higher level of confidence in the data if performed. Each node in the TDN is called a block and contains actions to be performed. A block also has goals, pre- and postcondition constraints and temporal constraints associated with it.

The TDN is used as a general representation of an operational sequence of tasks. An instance of a TDN is created from the general representation and parameterized for the specific track.[2] The TDN acts as a template for operations, and individual parameters (time, frequency, file names) are filled in at execution time to perform operations.

6.5 The Voyager TDN – specific context for the meeting

In this section we begin to describe the specific context for the knowledge acquisition meeting in which REBUS/SCtool was used. During the meeting, the software expert was to acquire and verify knowledge from a domain expert about a receiver subsystem called the telemetry processor (TP13). For background information, TP13 (also

[2]A track is a mission in which is a spacecraft or celestial body is being observed.

called arx2) was removed from the DSS-13 receiver in Goldstone, so the operators had little experience with it. The TP13 would be needed for the tracking of the Voyager Spacecraft, thus the software expert needed to interview the developer of the TP13 and the focus was on the Voyager TDN.

The knowledge of this TDN came from a trip to Goldstone by the software expert two days prior to the meeting. The purpose of this trip was to discuss the overall mission and to gather the knowledge which was later encoded in an initial TDN.[3] The TDN played a significant role in the meeting by focusing on candidate scenarios (corresponding to TDN blocks) which needed discussion with the TP13 domain expert.[4]

Figure 6.2 contains the TDN which was present in paper form during the meeting. The seven starred blocks are ones which were identified as needing discussion with a domain expert at the meeting.[5] The reason a block, instead of an entire TDN, was chosen as the unit of abstraction for mapping a scenario, was that a block corresponded to a manageable unit which might be re-used in other missions. Note also that the entire TDN contains conditionals and loops and alternative courses which also seem closer to *between*-scenario relationships.

To illustrate the TDN directives, figure 6.3 contains three blocks. The detail of the three blocks was entered/modified after the meeting. They are presented here for background. The block "Connect M&C to subsystems" has several commands to connect to various subsytems (e.g. "arx2@connect" is the command to connect to TP13). The block "Set Receiver(s) for Track Configuration" shows several commands to set attributes of TP13 (e.g. arx2@setvar@arxCarPredPwr1_D@15@Y is used to set the Pc/No - predicted carrier power). The block "Acquire Carrier" shows a manual action to be performed by the operator.

[3]I did not find out about this knowledge acquisition at Goldstone until the morning of the evaluation with the domain expert.

[4]The task of knowing which scenarios to write or on which to focus is highly dependent on domain knowledge and project objectives.

[5]The first six were identified by the software expert and the last by the domain expert during the meeting.

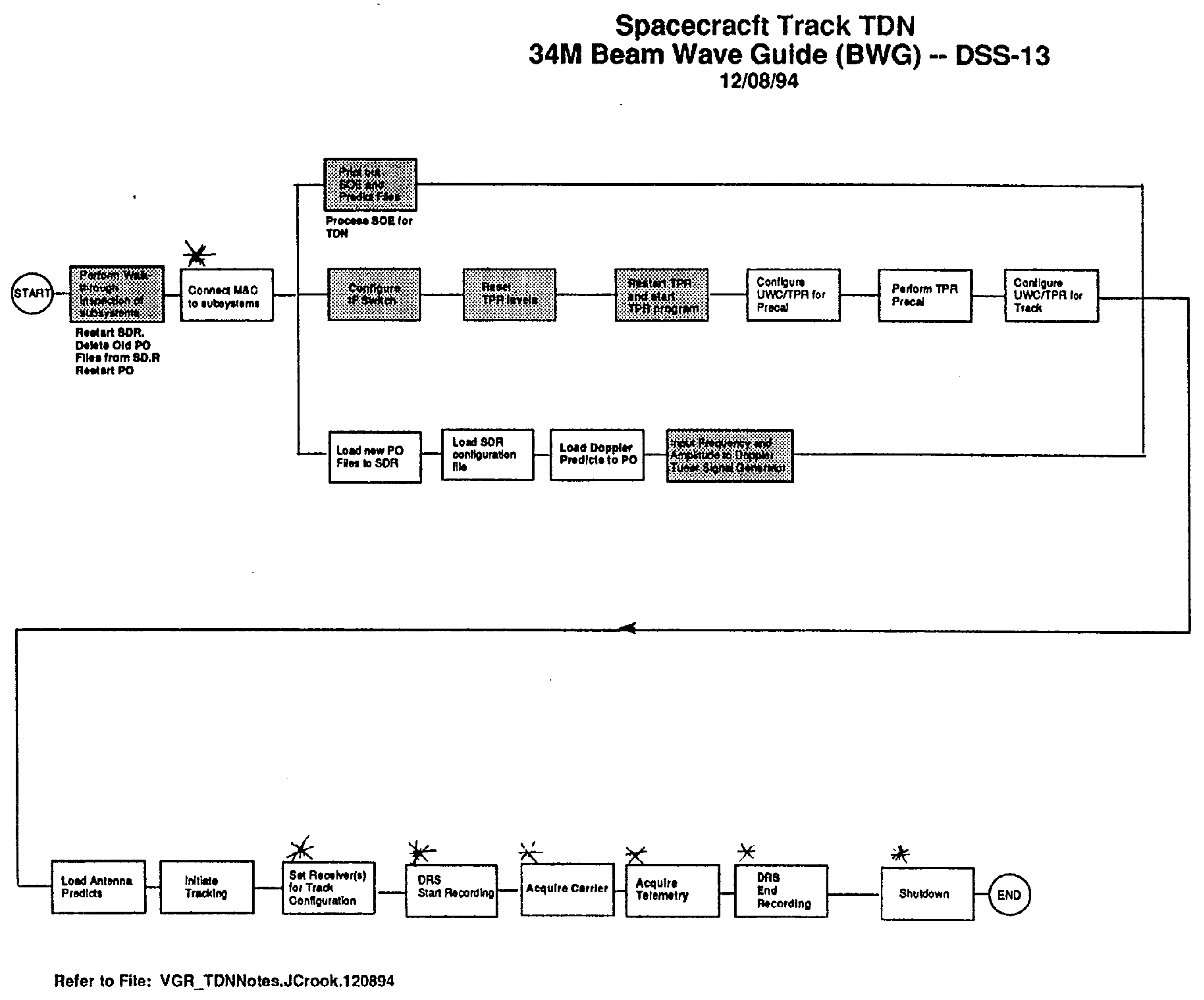

Spacecracft Track TDN
34M Beam Wave Guide (BWG) -- DSS-13
12/08/94
START
Perform Walk-through inspection of subsystems
Restart SDR. Delete Old PO Files from SD.R Restart PO
Connect M&C to subsystems
Print out SOE and Predict Files
Process SOE for TDN
Configure IF Switch
Reset TPR levels
Restart TPR and start TPR program
Configure UWC/TPR for Precal
Perform TPR Precal
Configure UWC/TPR for Track
Load new PO Files to SDR
Load SDR configuration file
Load Doppler Predicts to PO
Input Frequency and Amplitude to Doppler Tuner Signal Generator
Load Antenna Predicts
Initiate Tracking
Set Receiver(s) for Track Configuration
DRS Start Recording
Acquire Carrier
Acquire Telemetry
DRS End Recording
Shutdown
END
Refer to File: VGR_TDNNotes.JCrook.120894

Figure 6.2: The Voyager track TDN – used during the meeting (rotate page)

Connect M&C to subsystems

```
#Block 2
#
!SS_CONNECT
!COMMAND
!wx@connect
uwc34@connect
sdr@connect
ant34@connect
arx2@connect
#ifs@connect
!inspect_SS
!PRINT_TRACK_FILES
CONFIG_IF_SWITCH
LOAD_PO_FILES
!0
!NIL
!NIL
```

Set Receiver(s) for Track Configuration

```
#Block 15
#
!CONFIG_RCVR
!COMMAND
!arx2@setvar@arxCarPredPwr1_D@15@Y
arx2@setvar@arxCarPredFreq1_D@269000000@Y
arx2@setvar@arxCarBW1_D@0.5@Y
arx2@setvar@arxCarRate1_D@500@Y
arx2@setvar@arxCarLoop_S@2@Y
arx2@sendvar@arx2_config
!LOAD_ANT_PREDICTS
!CHK_TLM_DECODE
!0
!NIL
!NIL
```

Acquire Carrier

```
#Block 20
#
!ACQUIRE_CARRIER
!INPUT
!Type 'acquire carrier' at the Receiver;
!Press<cr> When Finished;
!START_RECORDING
!ACQUIRE_TLM
!0
!NIL
!NIL
```

Figure 6.3: Examples of blocks in Voyager TDN

6.6 The room, equipment, and participants

The meeting room (see figure 6.4) was a lab containing various workstations and equipment. SCtool was running on a Sun Workstation in the right corner of the room. It was equipped with two monitors and was running XVan, a virtual X server. This server made it possible to move SCtool windows between the monitors. This extra monitor was thought important given the limited screen space and the number of dialog boxes for SCtool's catalogs and editors. Despite this, early in the meeting, the 8.5x11 page which contained the Voyager TDN (described in section 6.2) was placed in front of the second monitor. The second monitor was used later in the meeting to move some of the dialogs out of direct focus. Trish stated that she didn't have much trouble managing the dialogs. Although, I noticed that SCtool needs support for keeping multiple editor dialog's (when open to the same concept) consistent.[6] This is an area for further implementation work.

The video camera used to record the meeting was placed high on a tripod, facing a workstation two meters away. There were four people present throughout the meeting: Trish, Roland, Richard, and Lorna. Trish was in a workstation chair in front of the primary monitor. She operated SCtool. Roland was seated to the right of Trish. Richard was behind and right of Roland, mostly out of camera view. Finally, Lorna was seated behind Trish, also mostly out of camera view.

The major participants' backgrounds and roles are as follows:

Trish is the software expert and the scenario writer using SCtool during the meeting. She holds a B.S. in computer science and has two years of DSN automation research support experience. She has done the knowledge acquisition for two other tracks (one, called KaAP, was used for practice with REBUS) which she encoded in RIDES and the TDN editor. Trish is the developer of the TDN editor. Except to arrange the meeting, she has not worked with Roland prior to the meeting.

[6]This occurred during the training/review meeting, but it did not occur during the meeting with the domain expert.

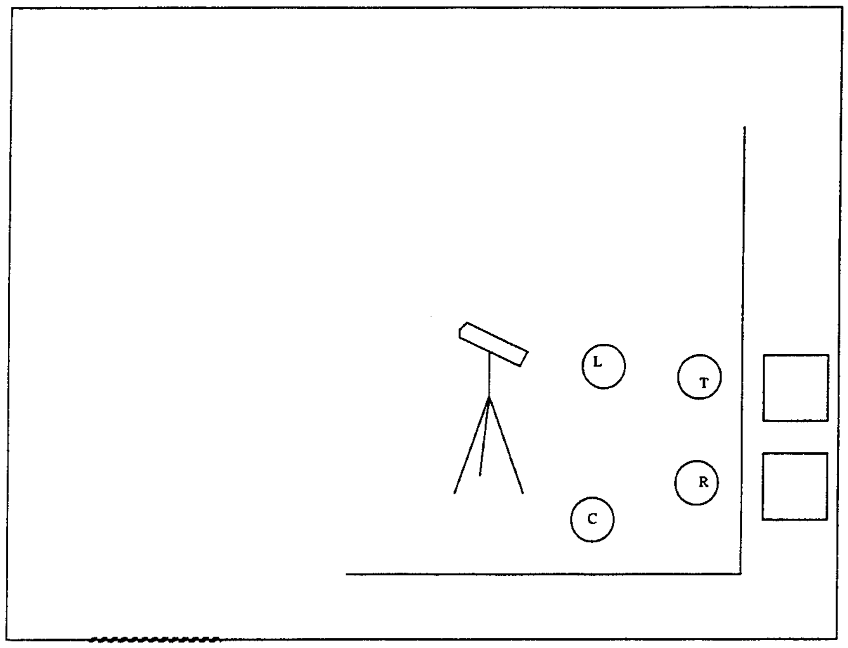


Figure 6.4: The meeting room (a lab)

Roland is the domain expert. He has "Approximately seven years experience working with prototype DSN receivers, testing with station equipment, and integration and testing with the research station (DSS-13)." He is the lead designer and developer of the telemetry processor (TP13). His educational background includes a M.S. in Electrical Engineering. He has computer programming experience.

Richard was asked to attend the meeting by Trish. He has worked with Roland before, has a M.S. in Computer Science, and six years of development experience with the DSN. Richard was free to ask and answer questions during the meeting.

Lorna is the developer of REBUS and SCtool.

The next section describes the background of the participants with REBUS and SCtool.

6.7 Training in REBUS and SCtool

Trish's training in REBUS/SCtool was informal, unguided and use-oriented. It seemed best to teach REBUS/SCtool in the course of trying to write scenarios (reflection-in-action). Thus, over the course of the case study, learning is occurring for both the student and the instructor. Trish was learning to write scenarios and to use SCtool and I was learning about the domain as well as about improvements which could be made to SCtool.

Three weeks prior to the meeting, SCtool was installed at JPL and Trish was given an informal demonstration. Over the course of a week Trish entered the KaAP TDN (about 4 hours of total time, over 3 days). She described 18 blocks of the KaAP TDN as scenarios. Also created were 12 objects and 10 units/types (5 list, 5 unit, 1 coordinate).

The following week I met with Trish to go over the scenarios and I attempted to address any problems. Two SCtool bugs needed to be fixed (both were known to randomly occur) and one simple feature was added (Trish requested a pushbutton to iconify the catalogs. This was in addition to the iconify button supplied by the Motif window manager). This meeting was videotaped and questions and issues raised are documented in section 6.12.1.

Richard and Roland were introduced to REBUS/SCtool during the first 13 minutes of the meeting. They were presented with hardcopies of the REBUS conceptual framework (as shown in figure 4.1) and a brief (9 min) demonstration of SCtool with data from the KaAP scenarios.

6.8 Timeline/background of JPL meetings

To clarify, several meetings are being referred to.

Set-up meeting A meeting in which SCtool was installed at JPL and then informally demonstrated to Trish. This occurred on Wednesday, November 23rd.

Training/review meeting A meeting to review Trish's KaAP scenarios and track the SCtool bugs. This occurred on November 30th and was videotaped. It lasted approximately 1.5 hours. Trish and Lorna were present. For this meeting, I performed most of the interaction with SCtool.

The meeting This is the meeting in which the REBUS/SCtool evaluation data was collected. It occurred on December 9th and was videotaped. The meeting was scheduled for 9:00am and started about 9:05am. In the beginning, Lorna sat at the console to explain REBUS/SCtool. After describing REBUS and SCtool, neither Roland nor Richard had questions, so Lorna exited SCtool. Trish then took control of the workstation. Then Trish began a new scenario corresponding to the block in figure 6.2 labeled Connect M&C to subsystems. This meeting was approximately two hours. At the end of the meeting Richard and Trish requested a copy of the videotape. I provided a copy to them at the follow-up meeting.

The follow-up/review meeting In this meeting Trish, Lorna, and Martin (a member of my dissertation committee and a requirements engineering researcher) reviewed the videotape. This meeting occurred on Dec 22.

6.9 Data captured in SCtool from the meeting

This section describes the data captured with SCtool for the Voyager track. Further analysis and discussion follows this section. For background, most of the following data was captured during the meeting. After the meeting, Trish continued to edit the scenarios adding five stimuli and associating them with scenarios. Trish also modified the list of "operator actions." What is presented here is the data after these modifications.

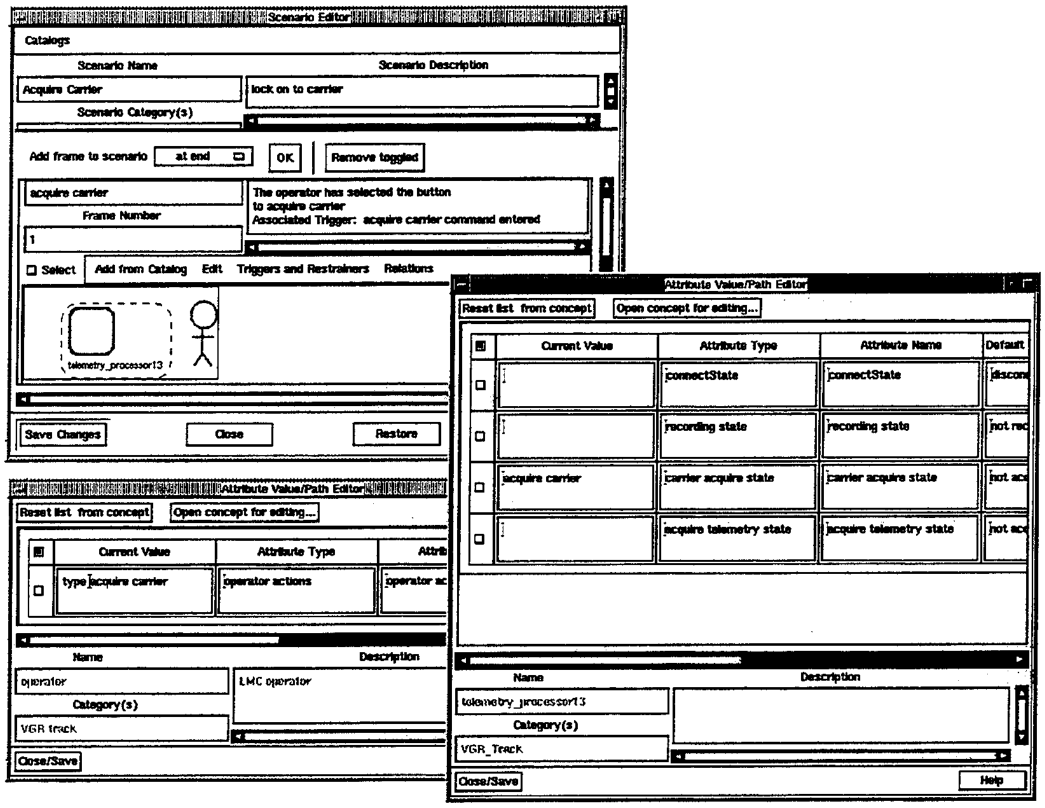
Figure 6.5 contains an example scenario captured during the meeting. The scenario is named "Acquire carrier" and it corresponds to the block of the TDN in figure 6.2 which is starred and labeled "Acquire Carrier." As stated in the scenario description, the operator needs to lock onto the carrier. In this scenario there are two objects, the telemetry_processor13 and the operator.

Inspecting the values of each object shows that, for the operator, the current value of the attribute "operator actions" is set to "type acquire carrier" and for the object, telemetry_processor13, the current value the attribute "carrier acquire state" is set to "acquire carrier." Trish has chosen to use a single frame to show more than one point in time. In this case she has actually presented causality within a frame. This could be considered an incorrect use of REBUS semantics (causality is between frames), but the circumstances of the evaluation don't lead us to believe this is a problem.

Specifically, Trish was asked to document any triggers and restrainers with the scenario frames' description due to SCtool limitations. She was not told that causality should be presented to the scenario reader *between frames* and this was not part of the REBUS language description she received prior to the meeting. Given she associated the causality of the trigger, "acquire carrier command entered" with the frame's description, this still conveys the correct notion. Further work is needed to investigate this issue of readability of scenarios written with causality in the frame versus between frames, since it is possible that both are equally understandable to scenario readers. Both also make explicit use of triggers and restrainers for semantics.

Another limitation of the prototype shows also shows up in this scenario. That is consistency checking. "Type acquire carrier" is meant to be the same as the actual item, "enter 'acquire carrier' command", in the list of operator actions. To provide automated support for consistency checking requires comparing the current attribute value against the attribute type and/or the path catalogs. Ideally, if the value entered for the attribute is not found then the system would recognize and track the inconsistency on a to-do list.

Figure 6.6 shows a unit of measurement "db-Hz" and a list type called "loop type" collected during the meeting. There are some interesting exchanges between the meeting participants about these types. For example, Richard said to Roland,



Scenario Name: Acquire Carrier

Scenario Description: lock on to carrier

Scenario Category(s):

acquire carrier

Frame Number: 1

The operator has selected the button
to acquire carrier
Associated Trigger: acquire carrier command entered

	Current Value	Attribute Type	Attrib
☐	type acquire carrier	operator actions	operator ac

Name: operator

Description: LMC operator

Category(s): VGR track

	Current Value	Attribute Type	Attribute Name	Default
☐		connectState	connectState	discon
☐		recording state	recording state	not rec
☐	acquire carrier	carrier acquire state	carrier acquire state	not ac
☐		acquire telemetry state	acquire telemetry state	not ac

Name: telemetry_processor13

Description:

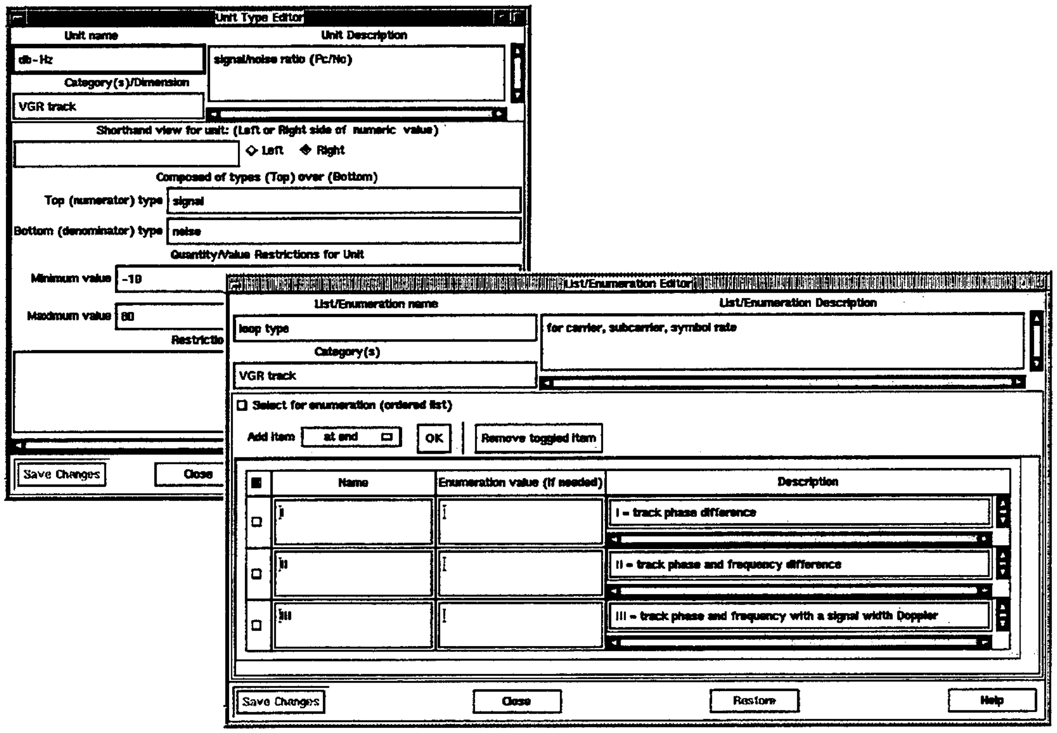
Category(s): VGR_Track

Figure 6.5: The scenario named "Acquire Carrier"

"I forget. What are the three types?" This occurred as Trish was editing the "loop type" list/enumeration dialog. Trish then proceeded to verify and correct the descriptive details of this dialog. The discussion of loop type is further described in section 6.12.2. The type "db-Hz" is significant to the revelation described in section 6.12.4.

Figure 6.7 shows the stimulus "acquire carrier command entered." It is associated with the scenario "Acquire carrier." As written by Trish, this dialog's left-side does not conform to what was expected. This issue is further analyzed in section 6.12.5.

To document the data captured with SCTool, figures 6.8, 6.9, 6.10, 6.11, and 6.12 contain the SCtool catalogs with all the concepts collected. I've also shown the catalog column "foid" which is the file identifier for each concept. This should not normally be shown as part of the user-interface to SCtool, but it is useful to show the foid details here because it is the time-stamp used to save the meeting data as files. A summary of the data and further discussion is in section 6.11.



Unit name: db-Hz

Unit Description: signal/noise ratio (Pc/No)

Category(s)/Dimension: VGR track

Shorthand view for unit: (Left or Right side of numeric value)

◇ Left ◆ Right

Composed of types (Top) over (Bottom)

Top (numerator) type: signal

Bottom (denominator) type: noise

Quantity/Value Restrictions for Unit

Minimum value: -10

Maximum value: 80

Restrictio

List/Enumeration name: loop type

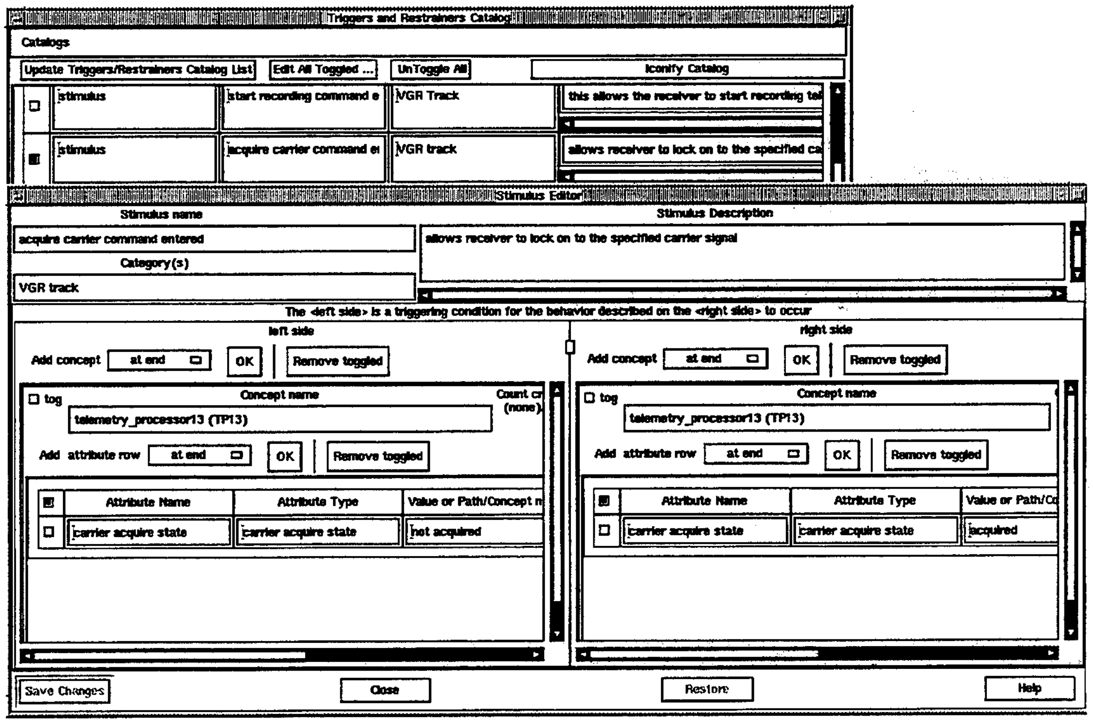
List/Enumeration Description: for carrier, subcarrier, symbol rate

Category(s): VGR track

☐ Select for enumeration (ordered list)

■	Name	Enumeration value (If needed)	Description
☐	I		I = track phase difference
☐	II		II = track phase and frequency difference
☐	III		III = track phase and frequency with a signal width Doppler

Figure 6.6: The unit, "db-Hz" and list/enumerated type named "loop type"



☐	stimulus	start recording command e	VGR Track	this allows the receiver to start recording tel
■	stimulus	acquire carrier command e	VGR track	allows receiver to lock on to the specified ca

Stimulus name: acquire carrier command entered

Stimulus Description: allows receiver to lock on to the specified carrier signal

Category(s): VGR track

The <left side> is a triggering condition for the behavior described on the <right side> to occur

left side

☐ tog Concept name: telemetry_processor13 (TP13) Count cr (none)

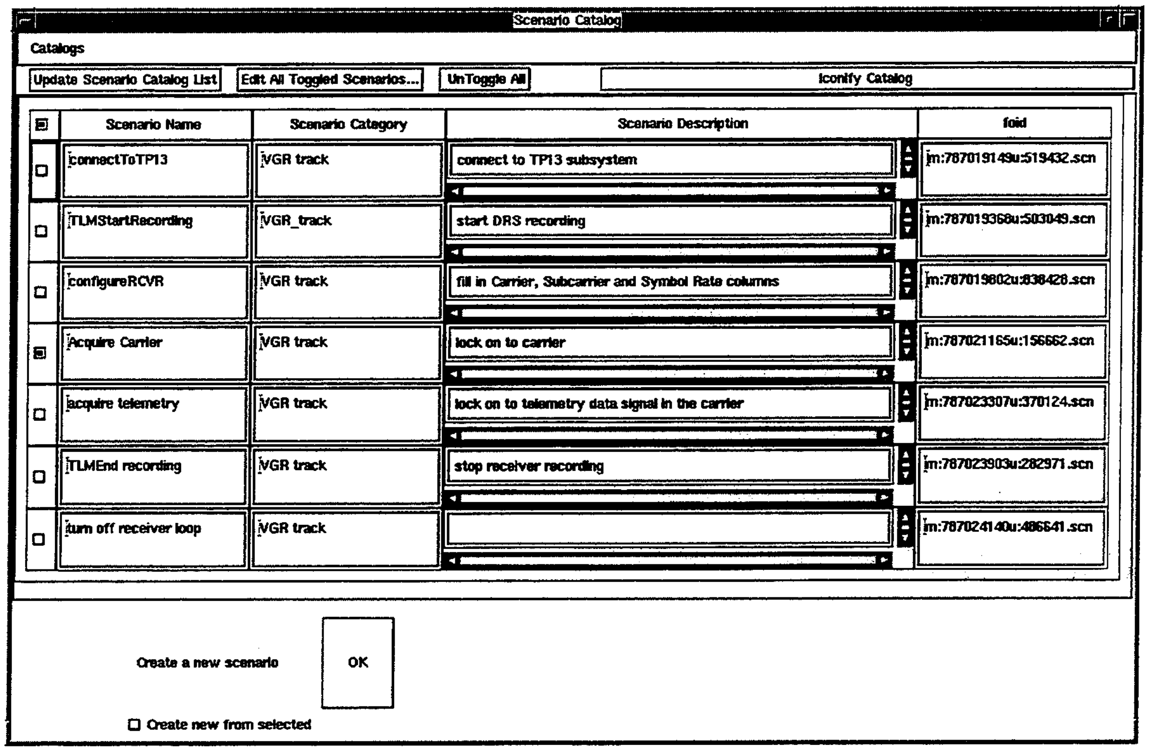
■	Attribute Name	Attribute Type	Value or Path/Concept n
☐	carrier acquire state	carrier acquire state	not acquired

right side

☐ tog Concept name: telemetry_processor13 (TP13)

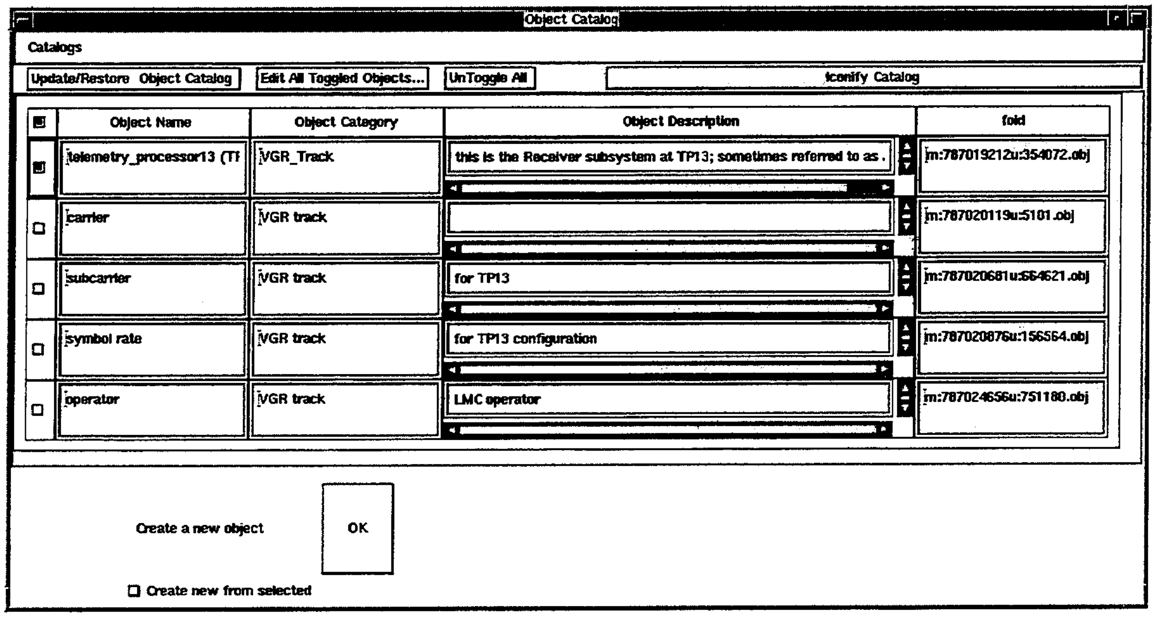
■	Attribute Name	Attribute Type	Value or Path/Co
☐	carrier acquire state	carrier acquire state	acquired

Figure 6.7: The stimulus "acquire carrier command entered"



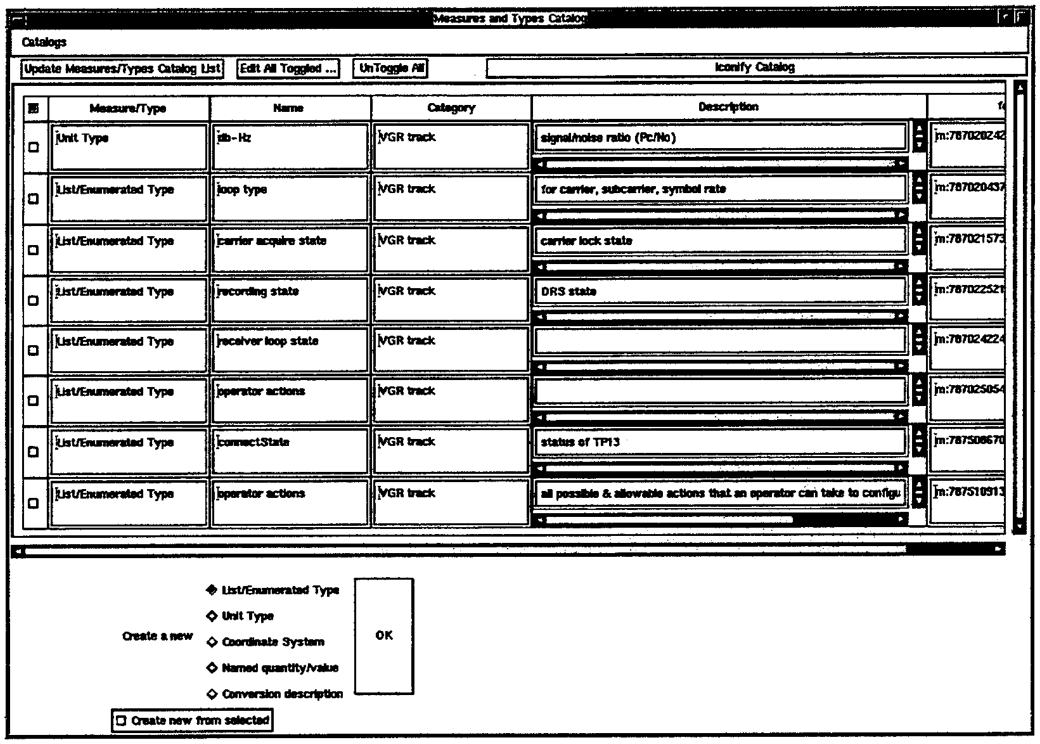
▣	Scenario Name	Scenario Category	Scenario Description	foid
□	connectToTP13	VGR track	connect to TP13 subsystem	m:787019149u:519432.scn
□	TLMStartRecording	VGR_track	start DRS recording	m:787019368u:503049.scn
□	configureRCVR	VGR track	fill in Carrier, Subcarrier and Symbol Rate columns	m:787019802u:838428.scn
▣	Acquire Carrier	VGR track	lock on to carrier	m:787021165u:156662.scn
□	acquire telemetry	VGR track	lock on to telemetry data signal in the carrier	m:787023307u:370124.scn
□	TLMEnd recording	VGR track	stop receiver recording	m:787023903u:282971.scn
□	turn off receiver loop	VGR track		m:787024140u:486641.scn

Figure 6.8: Scenario catalog (Voyager track data)



▣	Object Name	Object Category	Object Description	foid
▣	telemetry_processor13 (TI	VGR_Track	this is the Receiver subsystem at TP13; sometimes referred to as .	m:787019212u:354072.obj
□	carrier	VGR track		m:787020119u:5101.obj
□	subcarrier	VGR track	for TP13	m:787020681u:664621.obj
□	symbol rate	VGR track	for TP13 configuration	m:787020876u:156564.obj
□	operator	VGR track	LMC operator	m:787024656u:751180.obj

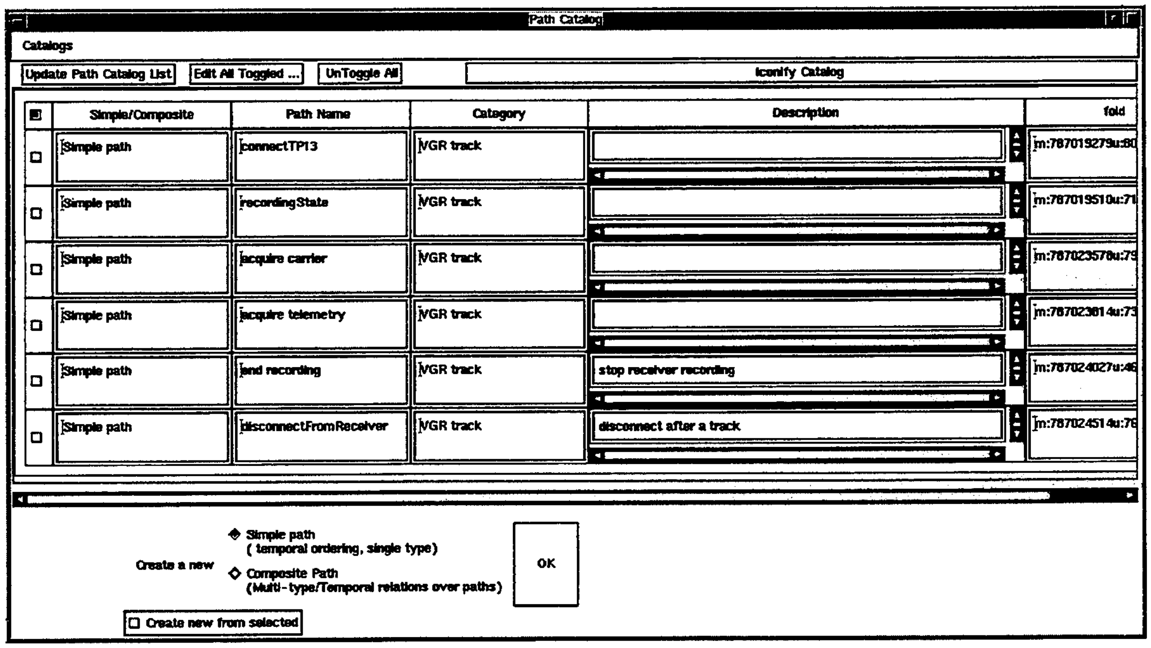
Figure 6.9: Object catalog (Voyager track data)


Measures and Types Catalog
Catalogs
Update Measures/Types Catalog List
Edit All Toggled ...
UnToggle All
Iconify Catalog

	Measure/Type	Name	Category	Description	f
☐	Unit Type	db-Hz	VGR track	signal/noise ratio (Pc/No)	m:787020242
☐	List/Enumerated Type	loop type	VGR track	for carrier, subcarrier, symbol rate	m:787020437
☐	List/Enumerated Type	carrier acquire state	VGR track	carrier lock state	m:787021573
☐	List/Enumerated Type	recording state	VGR track	DRS state	m:787022521
☐	List/Enumerated Type	receiver loop state	VGR track		m:787024224
☐	List/Enumerated Type	operator actions	VGR track		m:787025054
☐	List/Enumerated Type	connectState	VGR track	status of TP13	m:787508670
☐	List/Enumerated Type	operator actions	VGR track	all possible & allowable actions that an operator can take to configu	m:787510913

Create a new
◆ List/Enumerated Type
◇ Unit Type
◇ Coordinate System
◇ Named quantity/value
◇ Conversion description

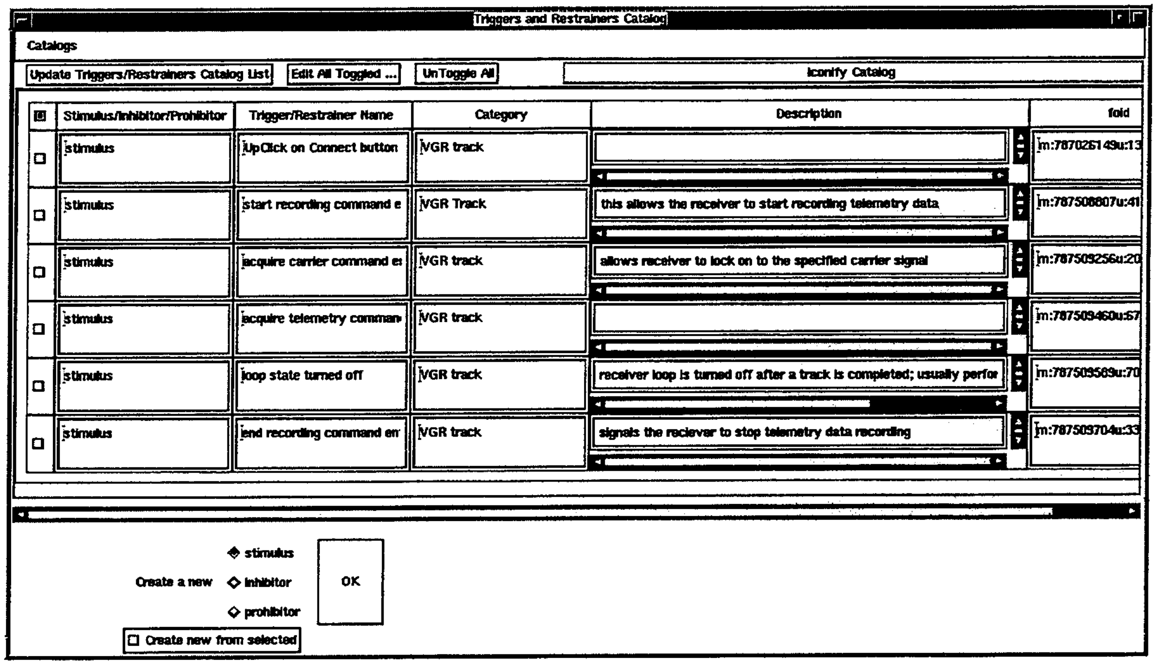
Figure 6.10: Measures/types catalog (Voyager track data)



	Simple/Composite	Path Name	Category	Description	fold
☐	Simple path	connectTP13	VGR track		m:787019279u:80
☐	Simple path	recordingState	VGR track		m:787019510u:71
☐	Simple path	acquire carrier	VGR track		m:787023578u:79
☐	Simple path	acquire telemetry	VGR track		m:787023814u:73
☐	Simple path	end recording	VGR track	stop receiver recording	m:787024027u:46
☐	Simple path	disconnectFromReceiver	VGR track	disconnect after a track	m:787024514u:76

Create a new
◆ Simple path
(temporal ordering, single type)
◇ Composite Path
(Multi-type/Temporal relations over paths)

Figure 6.11: Path catalog (Voyager track data)


Triggers and Restrainers Catalog
Catalogs
Update Triggers/Restrainers Catalog List
Edit All Toggled ...
UnToggle All
Iconify Catalog

☒	Stimulus/Inhibitor/Prohibitor	Trigger/Restrainer Name	Category	Description	foid
☐	stimulus	UpClick on Connect button	VGR track		m:787026149u:13
☐	stimulus	start recording command e	VGR Track	this allows the receiver to start recording telemetry data	m:787508807u:41
☐	stimulus	acquire carrier command e	VGR track	allows receiver to lock on to the specified carrier signal	m:787509256u:20
☐	stimulus	acquire telemetry comman	VGR track		m:787509460u:67
☐	stimulus	loop state turned off	VGR track	receiver loop is turned off after a track is completed; usually perfor	m:787509569u:70
☐	stimulus	end recording command en	VGR track	signals the reciever to stop telemetry data recording	m:787509704u:33

Create a new [✓ stimulus] inhibitor prohibitor
OK
☐ Create new from selected

Figure 6.12: Triggers and restrainers catalog (Voyager track data)

6.10 What else was captured - paper meeting notes

Trish took one page of paper notes during the meeting. The whiteboard was not used during the meeting. Trish later threw away her paper notes, but here is what she remembers them containing:

1. Options that we can take if we decide to decode telemetry data without a MCD (Maximum Likelihood Convolutional Decoder):

   a) acquire known data and compare received data with it

   b) decode off-line (maybe at DSS-14 where there is a MCD)

   c) use the DHT (Data Handling Terminal)

2. a graphical representation of the CONFIG_RCVR, CHK_TLM_DECODE, and START_TRACK blocks to indicate a change in block ordering

3. arx2 is TP13

In looking at the three items:

Item one describes three alternatives that were part of a "what-if" discussion that occurred during the meeting. This discussion was about the possibility that decoding would be done in addition to recording. The three items on the paper notes reflects the total persistent record (except for the video) about the MCD discussion. The discussion accounts for about thirty minutes of the meeting and some of the break time. During the break Trish explained these options to her project leader, Randy. Randy, verified these with Roland. Since the discussion was important enough to tell Randy, I asked Trish later about why she didn't document it with SCtool. Trish stated that she specifically didn't document these components because they would not be needed for the Voyager Track. Of course, it turned out that the discussion was important since decoding is now needed. A tool can not capture information that the tool user chooses not to enter or express.

The results of modifying the TDN to reflect item two is shown in figure 6.13. A conditional branch was added to the TDN and the original block was divided into two blocks.

Item three, arx2 is simply another name for TP13 and it is documented in the description of object TP13.

6.11 Data summary

The data collected is summarized by table 6.1. It shows the number and type of the SCtool building blocks that were collected. The building block concepts that were not used are: coordinate systems, conversions, composite paths, inhibitors, prohibitors, and all the spatial concepts. The fact that some building blocks were not used could be a sign that REBUS has too many concepts. However, it is more likely a result of the limits of this evaluation. On the positive side, it is an indication that more building blocks were not needed for this domain.

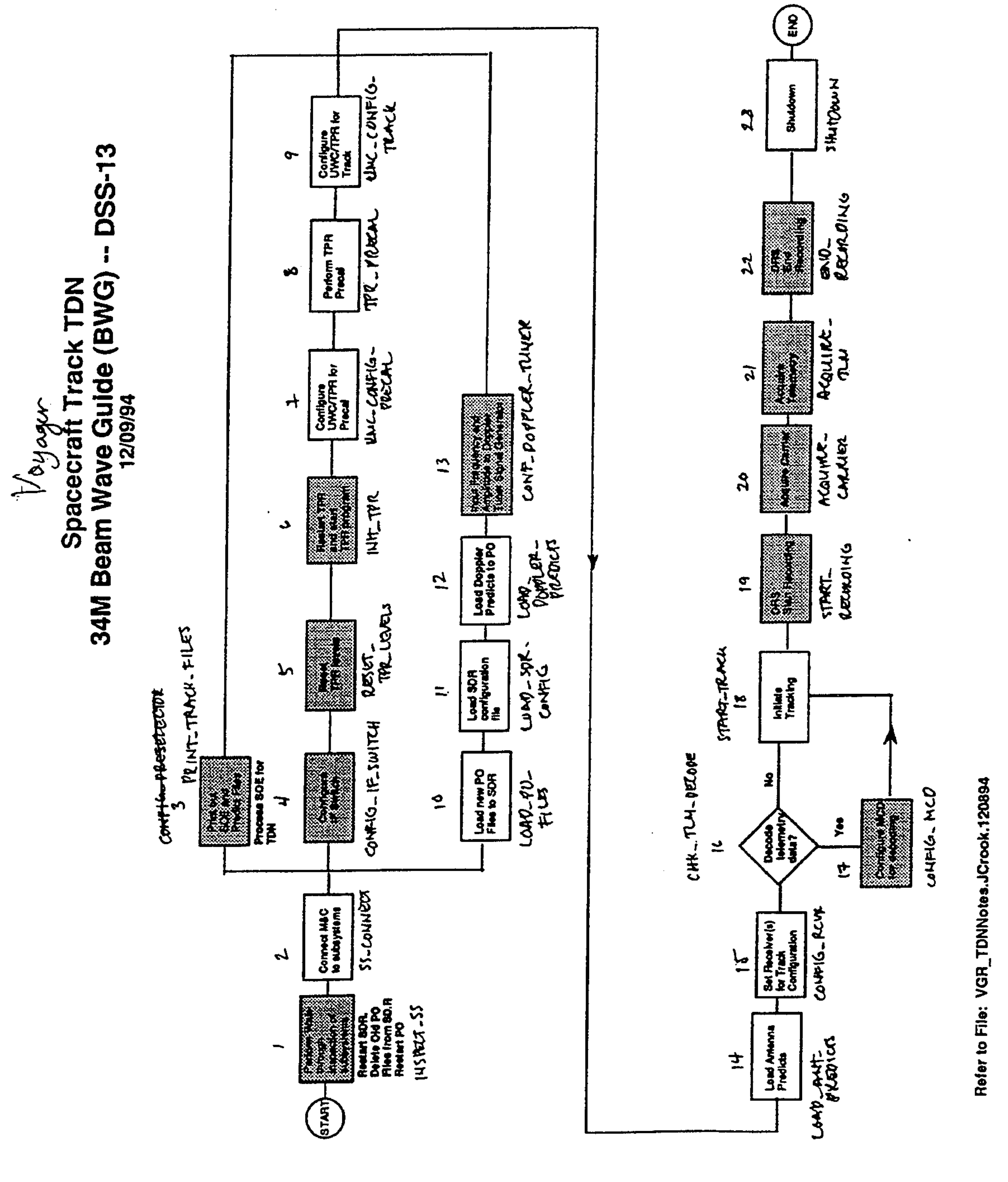


Figure 6.13: Voyager track TDN after modification (rotate page)

building block	*count*
scenarios	7[a]
objects	5
unit type	1
list/enumerated type	6[b]
simple path	6
stimulus	6
Total:	31

[a]These were all written as one frame with two or three objects.
[b]Ranging from 2 to 8 values (not included in count is a duplicate for "operator actions")

Table 6.1: Summary of building blocks collected during meeting

The flow of scenario creation is shown in table 6.2. The first column contains a scenario name followed by the name of the next scenario created (this corresponds to the TDN block order). The second column is the amount of time from the beginning of the first to the begining of the second scenario. The third column shows what building blocks were created after the first scenario was started and before the second was started. The data is based on the file names (foid's) SCtool used to save and restore concepts. The file name for each scenario is formed when the Scenario Editor Shell is created after selecting "new scenario". The filenames have two parts (m: - seconds u: - microseconds).

The average time between scenario creations was 13 minutes (79/6). Based on the videotape and this average, the meeting appears to be moving at a reasonable pace. Trish stated that her knowledge acquisition meetings are generally scheduled for about one hour. It is clear that it takes time to type in data and typing is slower than speech. The evidence of re-use of concepts across the scenarios suggests that it is certainly possible to take advantage of automated support for manipulation of the concepts found in scenarios. The next section contains further analysis.

scenario name / next scenario name	*time between creates*[a]	*building blocks created during*
connectToTP13 / TLMStartRecording	3 min.	(object) telemetry_processor13 (simple path) connectTP13
TLMStartRecording / configureRCVR	7 min.	(simple path) recordingState
configureRCVR / Acquire Carrier	22 min.	(object) carrier (unit type) db-Hz (list/enum) loop type (object) subcarrier (object) symbol rate
Acquire Carrier / acquire telemetry	35 min.[b]	(list/enum) carrier acquire state (list/enum) recording state
acquire telemetry / TLMEnd recording	9 min.	(simple path) acquire carrier (simple path) acquire telemetry
TLMEnd recording / turn off receiver loop	6 min.	(simple path) end recording
Total:	79 min.	

[a]Time is calculated as (next scenario file creation time - scenario file creation time) / 60 sec. [b]There was a morning-break of about 15 of the 35 minutes reported for the meeting.

Table 6.2: Time between scenario creations

6.12 Experience of evaluation and analysis

As part of the intelligent forces project, a videotape was made of a meeting in which the domain specific scenario acquisition tool (KBET) was used for the first time with a new pilot. The pilot used the whiteboard to explain new concepts which the tool could not easily capture: evidence that the domain-specific tool was too brittle for new knowledge. Furthermore, in trying to represent the new knowledge, the pilot had to discuss terminology used to build and represent new behaviors in the KBET. Relative to this, REBUS/SCtool was successful, since: (1) The whiteboard was not used during the meeting and what little knowledge that needed to be captured with pencil and paper was not the *within*-scenario domain knowledge needed to document the Voyager TDN. (2) During the meeting, no one raised questions about the REBUS/SCtool vocabulary or its use.

Although this indicates that SCtool was more successful than KBET, it's difficult to tell if the difference is a result of the tools themselves or the context in which they were used. It might be that SCtool worked better because the domain experts had more engineering background. But at least we can point to this as an encouraging result. The next sections provide more detailed examples and analysis.

6.12.1 Able to map domain knowledge to REBUS/SCtool

In this new domain, no new concepts needed to be added to REBUS for it to be used. While the DSN did not stress all the elements of REBUS, Trish did not have any difficulty in using the conceptual framework. The few indications of difficulty were minor and can be attributed to SCtool's prototype implementation, the building block editors, or *between*-scenario notions.

For example, instances of difficulty or questions that occurred in the training/review meeting are as follows:

1. Trish was modelling an object with three attributes, which were all objects of the same type. She gave each a unique name and a unique type. She asked

about this, and I clarified that one did not need a new type for every object attribute.

2. For fields in the list/enumerated types editor, there was some confusion as to which column should contain the items. As seen in figure 6.6, the first column is labeled "name" and the second "Enumeration value (if needed)." Changing the textual label to "element name" should improve this dialog.

3. Trish wanted to know if there was a way to delete or search for items from a catalog. This functionality was not implemented in the prototype.

4. At one point, Trish is engaging in what seems to be "thinking out loud." She is considering default values for the various predict points (planetary, sideal, and local) and states that the default values should be the last ones entered by the user. She seems to realize that this is not a true default and states, "if planetary has a value the others should be zero." I told her that what she wanted to express was a rule and I proceeded to model her statement as a stimulus.

During the actual evaluation meeting with the domain expert, Trish did not need to ask any modeling questions. She asked only one user interface question which was about the location of a scroll bar.

6.12.2 Domain knowledge captured and verified

Trish was able to capture and verify the domain knowledge she needed for the Voyager TDN. REBUS/SCtool were used in a process which was more than a simple review of a textual document, it also involved uncovering further details which are then captured. For example, when reviewing the attributes of the carrier object, Trish asks Roland, "Are there any default values?" While looking at the screen, Roland says, "For frequency, no; loop type there is an enumerated type. I, II, III. (I use Roman numerals)" Richard asks, "I forget what are the three types?" Roland explains in parallel with Trish creating a list/enumerated type named "loop type."

Roland and Richard proceed to clarify and verify the explanation Trish has recorded in the description field. The loop type can be seen in figure 6.6.

6.12.3 Achieved shared understanding

The videotape contains evidence that REBUS/SCtool was successfully used to achieve shared understanding between Trish and Roland. Relative to Richard's questioning of Roland during the meeting, there was much less reliance on speaking during the meeting to clarify understanding between Roland and Trish. The REBUS/SCtool dialogs focused the communication between Trish and Roland, so that Roland would just look at the screen, and point or gesture at the details that needed to be changed or discussed. A small example illustrating the value of having the shared external representation is relevant. In response to one of Richard's questions, Roland has lost the context of his conversation with Richard and asked "Where are we?" This did not occur with Trish's question.

6.12.4 Occurrence of side-scenarios

Side-scenarios were an unexpected part of the communication during the real-world use of REBUS/SCtool. A side-scenario occurred in an episode in which one of the attributes, Pc/No (the signal to noise ratio), of the carrier object was being defined. In this context, Trish created a unit type with Roland's feedback. Roland explained that the unit of measurement is db-Hz. In filling out the range of values for the dialog, Roland hesitated in stating the range of values as between -10 and 80. Sensing this, Lorna asked if it was the normal range. Roland's response was "As a unit there would be no real min or max for it, but that's sort of the range the receiver will expect to see and it doesn't operate outside that range. It's designed for that kind of range." Lorna then asked, "So what happens? Can someone enter a value outside the range." Roland responded with a side-scenario. Roland said, "You can enter a value outside the range. Since it's not designed at that value, for instance 80 db-Hz, the scaling can't accommodate a signal that strong. So, even if you had a signal that strong it wouldn't be operating properly. You would have internal overflow happening. So a signal less than -10 db-Hz is too weak for it to track."

Side-scenarios are close to being *between*-scenario relations. It's just that the situation in which this example arose was the context of defining a building block. This is more than the simple connection of a building block as participant in a scenario. It suggests a rational link between a concept and its side-scenarios.

6.12.5 Interpretation of triggers and restrainers

Figure 6.7 contains an example of a stimulus written by Trish. In looking at this dialog and the scenario "acquire telemetry" in figure 6.5, a difference of interpretation occurred. Specifically, one should be able to look at the objects in the scenario and the trigger and see how they relate causally. In connecting the <sides> to the phrase in the dialog, Trish's stimulus is read as: The <left side = TP13 is not acquired> is a triggering condition for the behavior described in the <right side = TP13 is acquired>. The causally oriented version would be written as: <left side = operator is enter 'acquire carrier' and TP13 is not acquired> <right side = TP13 is acquired>.

To explain this Trish states:

> I interpreted the left/right side as the state of the object *before* and *after* the occurrence of the stimulus. Stimulus name: acquire carrier command entered. State of TP13 before this action/stimulus occurred: carrier not acquired. State of TP13 after the action/stimulus occurred: carrier acquired.
>
> This parallelism between the two sides made more sense to me at the time I was learning how to use the triggers and restrainers editor. Also, I completely left out the operator because I didn't really care where the stimulus was coming from as the state of the object (tp13) is only dependent on the stimulus itself, not on the source of the stimulus.

The manner in which the stimulus was written does show causality, it just places semantic importance on the stimulus name. The alternative interpretation of how to write a stimulus could have resulted from the following three alternatives explanations.

1. Training/User Interface. I never explained to Trish that she should read a stimulus from left to right based on the text in the user interface.

2. The influence of the TDN pre- and post- conditions for a block. In the follow-up meeting Trish explained that the left side corresponded to the pre-condition for a block and the right side, the post-condition. So, Trish's background influenced her use of the dialog.

3. There needs to be a division between the internal and external system interface.

Alternative 3 has the most significant consequences for the interface. Other scenario notations, such as the message-flow diagram, make a distinction between external and internal events. In terms of changes to REBUS/SCtool, this would mean that triggers and restrainers would need to separate out external activity, such as the Operator's command, from the internal state.

6.12.6 Use of depictive abstraction

The scenarios collected for the DSN domain appear rather abstract compared to the IFOR scenarios. The DSN domain is a much more human designed domain than the air-combat scenarios which take place in the natural world, so the higher levels of abstraction may be more readily known. So, they appear to be sufficiently concrete for this domain.

To illustrate the use of abstraction, Trish's use of an object named "operator" indicates greater use of abstraction than had she used a real name for an operator. She also decided to use the SCtool abstract default depiction, the round-cornered boxes of the system component and its textual name (see the acquire carrier scenario in figure 6.5).

This can be explained by considering that it may have been a combination of the following alternative explanations:

1. As a software expert, Trish considered the role and the abstract system depiction sufficient. Also, given Roland's computer experience, more abstract depiction was sufficient.

2. The features of the task domain are naturally abstract (i.e. the domain depiction an expert would draw is a box).

3. The DSS-13 graphical user-interface was not running locally, so it was not easy to capture the concrete screens associated with a component.

4. Trish did not receive any training in evocative communication to encourage her to use concrete notions.

In considering explanations 1 and 2 for most of the meeting the abstract depiction was sufficient. Item 3 is important because at one point Trish asked Roland a question about the DSS-13 user-interface. She stated, "When I saw the UI there were two buttons. Does it matter which one you click-on?" Roland's response was, "I don't remember those screens, Richard do you remember?" Richard says, "No, I'll bring it up." Richard proceeded to bring up the system on another computer in the meeting room[7]. The task of setting up this program actually took about twenty minutes, since there were various problems in starting the software. While it is speculative to consider what would have happened had an evocative screen snapshot been readily available in SCtool, it is likely that Trish's question could have been answered as easily as it was asked.

The role of 3 and 4 in combination can't easily be determined from the one meeting. Trish had insufficient time before the meeting to even prepare scenarios a-priori much less capture screen snapshots. Before advocating further training to address explanation 4, as Pott's et al. point out [82], further research and experiments should be done on the role of concreteness in scenarios. Overall, SCtool has

[7]This computer was to the left of the workstation used for SCtool

the necessary features to support concrete and abstract depiction. For future work, automated support might analyze a collection of scenarios and provide some critique of a scenario collection which relied heavily on default depictions. Furthermore, the critic could post this to an agenda or "to-do" list mechanism.

6.13 Chapter summary

This chapter documents the experience of using REBUS and SCtool in the context of a real-world project at JPL. The evaluation was driven by the needs of JPL. This project provided an opportunity to formatively evaluate REBUS and SCtool. This was done by placing REBUS and SCtool in the context of a new application domain and letting potential users of scenario support tools evaluate REBUS/SCtool for their needs.

In the context of a meeting which was videotaped, the software expert used SCtool to collaborate with a domain expert. The software expert was involved in acquiring domain knowledge from the system engineer about a particular system object, called telemetry processor 13, and how it would be used in the context of a particular mission to track the Voyager spacecraft.

The domain knowledge collected during the meeting is summarized and presented. Within the limitations of the prototype, REBUS and SCtool were found to be useful for gathering domain knowledge during the meeting. REBUS provided sufficient conceptual coverage for the concepts needed in the new domain, although some REBUS concepts, such as the spatial concepts, were not needed for the evaluation.

The evaluation highlighted the need to develop SCtool further. Support for organizing scenarios, searching for concepts, and checking consistency are needed in the next version of the prototype.

Chapter 7

Conclusion and future work

This chapter highlights the accomplishments of this dissertation and illuminates possibilities for future research.

7.1 Summary of dissertation

Scenarios are one of the most natural methods of communicating domain knowledge between domain and software experts. Based on an observational study of scenarios communicated between such people and a literature survey, I found the existing scenario representations inadequate for precisely describing the richness of the domain knowledge contained in the natural scenarios written by domain experts. The natural scenarios illustrated multiple objects engaged in simultaneous behavior. They contained a mix of depictions, descriptions, objects, units of measurement and types, spatial concepts, temporal concepts, and behavioral concepts. Existing conceptual frameworks either provided too little coverage or far too many categories with which to classify domain knowledge.

In surveying potential tools to provide automated support for scenarios, what existed was found to be inadequate in light of the use of scenarios for precise communication. When scenarios are captured in the current practice, scenario writers are primarily using text editors and drawing tools to capture and manipulate the domain knowledge. Such tools do not specifically focus on the scenario writing task and do not specifically support the semantic definition and manipulation of the acquired domain knowledge. Alternatively, tools exist for the semantic definition and manipulation of domain knowledge but they are focused on the details needed for the software domain and don't relate back to the application domain.

For this dissertation, a new representation (REBUS) for scenarios was developed and embodied in a automated tool (SCtool). REBUS was designed to support the richness of the domain knowledge contained in the natural scenarios written by the domain experts. This coverage was demonstrated: by examples from various domains; by considering the requirements for a expressive scenario representation; and by an evaluation in an application which was not considered prior to REBUS's development. The design of REBUS and the graphical user-interface to SCtool considered how concepts could be organized within a conceptual framework and as building blocks. The conceptual framework served as a basis for the tool's catalogs and each building block was provided as an editor.

In evaluating REBUS and SCtool in a new domain, REBUS was able to provide the support for the conceptual coverage, the depiction and description, and the multiple object behavior. All of which are needed by the scenario writer. Although the domain was not highly spatial, the whiteboard was not needed during the meeting and the personal notes that were taken were focused on a topic which was tangential to the scenarios and concepts which the scenario writer wanted to model. Compared to text and drawing tools, SCtool's output contained a more precise record of domain concepts identified in the meeting. The domain concepts were captured in a structured manner at an appropriate semantic level of abstraction. During the course of the evaluation, SCtool was easily used to create new scenarios, objects, units of measurement, and paths. Only one question was asked about the user-interface, specifically, concerning the location of a scroll bar. The domain independence of REBUS/SCtool was not a hindrance. The participants were able to actively engage in modeling their domain and they were not distracted or confused by the terminology of REBUS.

REBUS/SCtool was evaluated in the domain of operational control of the Deep Space Network (DSN). It was chosen to be different from the Intelligent Forces (IFOR) domain which served as initial motivation for REBUS' design. While success in this different domain shows the domain-independence of REBUS, it does not serve to demonstrate that REBUS is suited to *any* domain for *any* domain experts. It would be great to try SCtool back in the IFOR domain, but this work is not part of this thesis.

In light of this, the evaluation in this thesis could be thought of as a pilot study to show the potential effectiveness of REBUS and SCtool. Further development and evaluation should follow to characterize the range of tasks and range of expertise for which a tool such as REBUS/SCtool is suited. This is best started after *between*-scenario support is implemented.

The DSN study was done with experts who have more systems background than the pilots. The required domain knowledge stressed only a part of the REBUS conceptual framework, namely objects, units/types, simple paths, and triggers. The fact that other aspects of REBUS were not stressed, does not indicate that they are not needed.

As with any prototype system, SCtool falls short of an ideal implementation. Many details were not implemented due to lack of time. To implement these details one would need a robust, object-oriented database and graphical interface framework.

7.2 Future work

There are several directions for further research. We need to continue to improve REBUS and SCtool based on the weaknesses identified in chapters 5 and 6. Such as providing facilities to query and retrieve concepts and improvements to the path editors, and triggers and restrainers building blocks. The use of REBUS and SCtool for real work highlighted the need to implement *between*-scenario support (sec 7.2.1). In addition SCtool can be extended in several directions (sec 7.2.2-7.2.6).

7.2.1 *Between*-scenario support

Section 3.4 described many of the relations potentially needed for between scenario support. Further work is needed to investigate how such relations are realized and used in an automated tool. One interesting direction from the air-combat domain is shown in figure 7.1.[1] As well as showing scenario composition and conveying temporal order and disjunction, the figure illustrates a more abstract scenario which still contains concrete detail (e.g. "The target turns 180° and runs") and informal annotation, such as the comment "this is trivial" in box BDT 8b2a.

7.2.2 Connection to automated tools for software experts

We believe that the REBUS building blocks are transformable to and from an automated knowledge based software development environment. Consider a tool for software experts which could accept REBUS as input or output. That is, if domain knowledge modeled with REBUS is input, the system provides the user with a graphical user interface to transform building blocks to programming-specific knowledge. If programming-specific knowledge is input, the tool could provide the user with the ability to associate that knowledge to REBUS concepts. The tool would support traceability by maintaining the mapping between scenarios and detailed system specifications. For example, a unit of measurement like nautical miles could be transformed to an appropriate "class" or a programming type like "integer" could be mapped back to nautical miles.

7.2.3 Automated support for multiple users

To support multiple scenario readers and writers on a project both synchronous and asynchronous automated support can be used. A non-intrusive mechanism for the synchronous editing of the scenarios is the replication of the interface dialogs to multiple workstations for simultaneous viewing and editing. To do this in an X Window System environment, one could use a commercial tool such as HP's

[1]Figure 2.4 contains the page corresponding to the box labeled 9b2b.

Navigation
&
Conventions

1. "Scene" 1a1a = standard initial conditions; 3 x 2 ship elements on CAP vs 2 x enemy fighters @ 100 nm range.
2. Target assignment and commit decision are presumed.
3. Branches occur at significant decison points.
4. The "a1a" trace represents the stereotype version. (desired, simplest outcome)
5. Traces (currently) conclude at a "high Pk" launch against the target.

"Bagdad Taxi Drill" tactic map

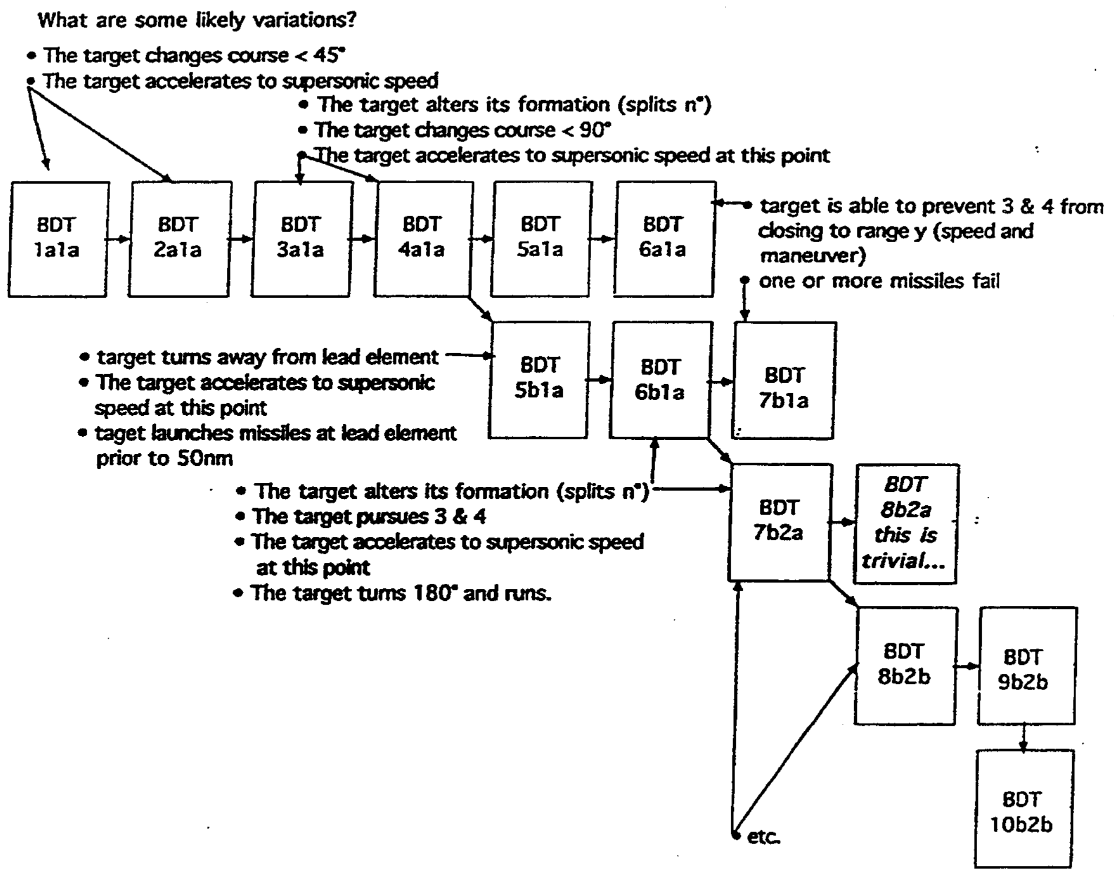

What are some likely variations?
• The target changes course < 45°
• The target accelerates to supersonic speed
• The target alters its formation (splits n°)
• The target changes course < 90°
• The target accelerates to supersonic speed at this point
BDT 1a1a
BDT 2a1a
BDT 3a1a
BDT 4a1a
BDT 5a1a
BDT 6a1a
target is able to prevent 3 & 4 from closing to range y (speed and maneuver)
one or more missiles fail
• target turns away from lead element
• The target accelerates to supersonic speed at this point
• taget launches missiles at lead element prior to 50nm
BDT 5b1a
BDT 6b1a
BDT 7b1a
• The target alters its formation (splits n°)
• The target pursues 3 & 4
• The target accelerates to supersonic speed at this point
• The target turns 180° and runs.
BDT 7b2a
BDT 8b2a this is trivial...
BDT 8b2b
BDT 9b2b
BDT 10b2b
etc.

Figure 7.1: Example of scenario composition from fighter-plane domain.

SharedX. Tool-specific synchronization support can also occur at the user-interface to provide visual context for users sharing scenarios. At the knowledge-base level, one needs to prevent inconsistent changes to the knowledge-base.

For asynchronous support, we could consider tools such as Lotus Notes^{TM} which tie together and filter semi-structured messages sent via electronic mail. Scenarios could be sent via e-mail for comment. Thus, the work-flow could be monitored, leading to automated support for agendas or "to-do" lists.

7.2.4 Automated support for agendas

Agenda mechanisms are another means to support individuals or groups. Scenarios or building blocks that are incomplete or that have open questions could be documented and tracked in a meaningful fashion. That is, agenda items could have links to building blocks, scenarios, or between-scenario concepts.

7.2.5 Automated support for variation generation

REBUS lends itself to automated support for the generation of scenario or building-block variants. Because of the conceptual framework's strong semantics, one can generate interesting variations which differ along one or more semantic dimensions. For example, variations on the temporal situation could be generated by varying the duration specifications of paths or varying the frame order. Some criteria for "interesting" scenario variations could be developed as a means to support analysis and validation of the system under development.

7.2.6 REBUS as a query language

Domain descriptions expressed in terms of REBUS could be used as a language to allow users to query and retrieve information from multimedia knowledge bases. The key issues here include the mapping between REBUS and the database schema and an appropriate graphical user interface.

Reference List

[1] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Lowell Jay Arthur. *Rapid Evolutionary Development: Requirements, Prototyping & Software Creation.* John Wiley & Sons, Inc., New York, 1992.

[3] Lon Barfield, Willie van Burgsteden, Ruud Lanfermeijer, Bert Mulder, Jurriënne Ossewold, Dick Rijken, and Philippe Wegner. Interaction Design at the Utrecht School of the Arts. *SIGCHI Bulletin*, 26(3):49–86, July 1994.

[4] Brigham Bell. *Using Programming Walkthroughs to Design a Visual Language.* PhD thesis, University of Colorado, Boulder, CO, 1992.

[5] Kevin M. Benner. The ARIES Simulation Component (ASC). In *Proceedings of the Eighth Knowledge-Based Software Engineering Conference (KBSE'93)*, pages 40–49. IEEE, September 1993.

[6] Kevin M. Benner. *Validation of Formal Specifications via Simplification and Simulation.* PhD thesis, University of Southern California, 1995.

[7] Jacques Bertin. *Semiology of Graphics.* Univerity of Wisconsin Press, Madison, Wisconsin, 1983.

[8] Mark Boddy. AAAI-92 Workshop Report: Implementing Temporal Reasoning. *SIGART Bulletin*, 4(3):15–49, 1992. Collection of workshop papers.

[9] Barry W. Boehm and Rony Ross. Theory-W Software Project Management: Principles and Examples. *IEEE Transactions on Software Engineering*, 15(7):902–916, July 1989.

[10] Grady Booch, editor. *Object Oriented Design with Applications.* Benjamin/Cummings, Menlo Park, CA, 1991.

[11] Dave Bridgeland. *Simulacrum: A System Behavior Example Editor*, chapter 10, pages 191–202. Volume 1 of Ichikawa et al. [46], 1990.

[12] Lee Brownston, Robert Farrell, Elaine Kant, and Nancy Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming.* Addison-Wesley Publishing Company, 1985.

[13] J.M. Carroll and M.B. Rosson. Getting Around the Task-Artifact Cycle: How To Make Claims and Design by Scenario. *ACM Transactions on Information Systems*, 10(2):181–212, April 1992.

[14] John M. Carroll. Names and Naming: An Interdisciplinary Review. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., October 1978.

[15] John M. Carroll. Natural Strategies in Naming. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., February 1979.

[16] Message Sequence Charts. International Telecommunication Standard CCITT Z.120 (1993).

[17] H.H. Clark and S.E. Brennan. Grounding in Communication. In L. Resnick, J. Levine, and S. Teasley, editors, *Perspectives on Socially Shared Cognition.* American Psychological Association, 1991.

[18] Herb Cohen. *You Can Negotiate Anything.* Bantam Books, 1982, c1980.

[19] J. Conklin and M.L. Begeman. gIBIS—A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, 1988.

[20] The X Consortium. The X Window System. Available via anonymous FTP at ftp.x.org.

[21] Jacques Davy. *GoPATH Programmer's Guide.* Bull - Imaging & Office Solutions, Paris, France, 1.2 edition, 1992.

[22] Jacques Davy. *GoPATH Reference Manual.* Bull - Imaging & Office Solutions, Paris, France, 1.2 edition, 1992.

[23] Sarah A. Douglas, David Novick, and Russell S. Tomlin. Consistency and Variation in Spatial Reference. In *Proceedings of the Ninth International Conference on Cognitive Science*, July 1987.

[24] Joseph L. Dvorak and Thomas G. Moher. A Feasibility Study of Early Class Hierarchy Construction in Object-Oriented Development. *Empirical Studies of Programmers*, 4:23–35, 1991.

[25] W. L. Johnson et al. Collected papers of the SOAR/IFOR project. Technical Report ISI/SR-94-367, USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, Spring 1994.

[26] Kristina Fayyad and Lynne Cooper. Representing Operations Procedures using Temporal Dependency Networks. In *Proceedings of the Second International Symposium on Ground Data Systems for Space Mission Operations, SPACEOPS-92*, Pasadena, CA, November 1992.

[27] William Finzer and Laura Gould. Programming by Rehearsal. *Byte*, 9(6):187–210, June 1984.

[28] Nick V. Flor and Edwin L. Hutchins. Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance. *Empirical Studies of Programmers*, 4:36–64, 1991.

[29] David H. Gelernter and Suresh Jagannathan. *Programming Linguistics.* MIT Press, 1990.

[30] Vinod Goel. "Ill-Structured Representations" for Ill-Structured Problems. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 130–135, 1992.

[31] Vinod Goel and Peter Pirolli. Motivating the Notion of Generic Design within Information Processing Theory: The Design Problem Space. *AI Magazine*, Spring:19–36, 1989.

[32] Neal Goldstein and Jeff Alger. *Developing Object-Oriented Software for the Macintosh: Analysis, Design, and Programming.* Addison-Wesley Publishing Company, Reading, MA, 1992.

[33] Reginald G. Golledge. Do People Understand Spatial Concepts: The Case of First-Order Primitives. In *Theories and Methods of Spatio-Temporal Reasoning in Geographics Space*, pages 1–21. Springer-Verlag, 1992. Lecture Notes in Computer Science 639.

[34] S. Gossain and B. Anderson. An Iterative-Design Model for Reusable Object Oriented Software. In *Proceedings of OOPSLA ECOOP '90: Conference on Object-oriented Programming: Systems, Languages, and Applications, European Conference on Object-oriented Programming*, New York, NY, October 1990. ACM Press.

[35] R.V. Guha and D.B. Lenat. Cyc: A midterm report. *AI Magazine*, 11(3):32–59, 1991.

[36] Raymonde Guindon. Designing the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, 5:305–344, 1990.

[37] Raymonde Guindon. Knowledge Exploited by Experts During Software System Design. *International Journal of Man-Machine Studies*, 33:279–304, 1990.

[38] Raymonde Guindon and Bill Curtis. Control of Cognitive Processes During Software Design: What Tools Are Needed? In *Proceedings of the Conference on Human Factors in Computing Systems(CHI)*, pages 263–268. ACM, May 1988.

[39] Robert J. Hall. Interactive Specification Acquisition via Scenarios: A Proposal. Technical report, AT&T Bell Laboratories, September 1992.

[40] Robert J. Hall. Validation of Rule-Based Reactive Systems by Sound Scenario Generalization. In *Proceedings of the Eighth Knowledge-Based Software Engineering Conference (KBSE'93)*, pages 30–39. IEEE, September 1993.

[41] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[42] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–413, April 1990.

[43] Randall W. Hill, Kristina Fayyad, Patricia Santos, and Kathryn Sturdevant. Knowledge Acquisition and Reactive Planning for the Deep Space Network. Appeared in AAAI Fall Symposium on Planning and Learning: On to Real Applications, November 1994.

[44] Pei Hsia, Jayarajan Samuel, Jerry Gao, David Kung, Yasufumi Toyoshima, and Chris Chen. Formal Approach to Scenario Analysis. *IEEE Software*, pages 33–41, March 1994.

[45] Valerio Hunt and Andres Zellweger. The FAA's Advanced Automation System: Strategies for Future Air Traffic Control Systems. *IEEE Computer*, 20(2):19–32, February 1987.

[46] Tadeo Ichikawa, Erland Jungert, and Robert R. Korfhage, editors. *Visual Languages and Applications*. Plenum Press, New York, NY, 1990.

[47] Neil A. Iscoe. *Domain-Specific Programming: An Object-Oriented and Knowledge-Based Approach To Specification and Generation.* PhD thesis, The University of Texas at Austin, December 1990. Department of Computer Science.

[48] ISX for DARPA-Rome Laboratory. *Knowledge Representation specification Language (KRSL)*, version 2.0.1 edition, 1992. DRAFT of KRSL Standard for DARPA/Rome Laboratory Planning and Scheduling Initiative.

[49] I. Jacobson, M. Christenson, P. Johsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* ACM Press, 1992.

[50] W. Lewis Johnson. personal communication.

[51] W.L. Johnson and M.S. Feather. Requirements Analysis Using ARIES: Themes and Examples. In *Proceedings of the 5th Knowledge Based Software Engineering Conference, tech report no. RL-TR-91-11, Rome Laboratory*, pages 121–131, Syracuse, NY, September 1990.

[52] W.L. Johnson, M.S. Feather, and D.R. Harris. Representation and Presentation of Requirements Knowledge. *IEEE Transactions on Software Engineering*, 18(10):853–869, October 1992.

[53] Ken Kahn. ToonTalk^{TM} – An Animated Programming Environment for Children. Available via anonymous ftp from csli.stanford.edu in ftp/pub/Preprints/toontalk.ps.Z, February 1995. Animated Programs, 44 El Rey Road, Portola Valley, CA 94028.

[54] John Karat and John Bennett. CSCW '92 Workshop Report: Understanding and Supporting Successful Group Work in Software Design. *SIGCHI Bulletin*, 25(4), October 1993.

[55] John Karat and John L. Bennett. Using Scenarios in Design Meetings - A Case Study Example. In *Taking Software Design Seriously: Practical Techniques for Human-Computer Interaction Design*, pages 63–94. Academic Press, 1991.

[56] David Keirsey, Jimmy Krozel, David Payton, and David Tseng. Case-Based Computer Generated Forces. In *Proceedings of the Fourth Conference on Computer Generated Forces*, volume 4, pages 307–316, Orlando, Florida, 1994. Institute for Simulation and Training, Univ. of Central Florida. IST-TR-94-12.

[57] Van E. Kelly and Uwe Nonnenmann. Reducing the complexity of formal specification acquisition. In *Automating Software Design*, pages 41–64. AAAI Press, 1991.

[58] Kari Kuutti. Activity Theory and Its Applications to Information Systems Research and Development. In H.-E. Nissen, editor, *Information Systems Research*, pages 529–549. Elsevier Science, 1991.

[59] George Lakoff. *Women, Fire and Dangerous Things: What Categories Reveal About the Mind.* Univ. Chicago Press, Chicago, 1987.

[60] Barbara Landau and Ray Jackendoff. "What" and "Where" in Spatial Language and Spatial Cognition. *Behavioral and Brain Sciences*, 16(2), 1993.

[61] Allen MacLean, R. Young, V. Bellotti, and Thomas Moran. Questions, Options, and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction*, 6(3&4):201–250, 1991. Special Issue on Design Rationale.

[62] Kim Halskov Madsen. A Guide to Metaphorical Design. *Communications of the ACM*, 37(12):57–62, December 1994.

[63] C.M.I.M. Matthiessen and J.A. Bateman. *Systemic-Functional Linguistics in Language Generation: Penman.* Academic Press, 1991.

[64] G.A. Miller. The Magic Number Seven Plus or Minus Two: Some Limits on Our Capacity for Information Processing. *Psychological Review*, 63(2):81–96, 1956.

[65] A. Munro, M.C. Johnson, D.S. Surmon, and J.L. Wogulis. Attribute-Centered Simulation Authoring for Instruction. In *AI-ED 93, World Conference on Artificial Intelligence in Education*, Edinbugh, Scotland; 23-27 August 1993.

[66] Bonnie A. Nardi. Studying Context: A Comparison of Activity Theory, Situated Action Models, and Distibuted Cognition. In *East-West International Conference on Human-Computer Interaction: Proceedings of the EWHCI'92*, pages 352–359, 1992.

[67] Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End-User Computing.* MIT Press, Cambridge, MA, 1993.

[68] Bonnie A. Nardi and James R. Miller. Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development. *International Journal of Man-Machine Studies*, 34(2):161–184, 1991.

[69] Jean-Marc Nerson. Applying Object-Oriented Analysis and Design. *Communications of the ACM*, 35(9):63–74, September 1992.

[70] Uwe Nonnenmann and John K. Eddy. KITSS – Toward Software Design and Testing Integration. In *Proceedings of the AAAI-91 Workshop on Automating Software Design: Interactive Design*, pages 131–137, July 1991.

[71] Uwe Nonnenmann and John K. Eddy. KITSS – A Functional Software Testing System Using a Hybrid Domain Model. In *Proceedings of the Eighth IEEE Conference on Artificial Intelligence Applications.* IEEE, July 1992.

[72] National Institute of Standards and Technology. A Brief History of Measurement Systems with a Chart of the Modernized Metric System. poster, 1991. Special publication 304A.

[73] Greg R. Olsen and Thomas R. Gruber. Ontolingua Theory: physical-quantities. World Wide Web. http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/physical-quantities/index.html.

[74] Greg R. Olsen, Thomas R. Gruber, and Yves Peligry. Ontolingua Theory: standard-units. World Wide Web. http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/standard-units/index.html.

[75] Gary M. Olson, Judith S. Olson, Mark R. Carter, and Marianne Storrøsten. Small Group Design Meetings: An Analysis of Collaboration. *Human-Computer Interaction*, 7(4):347–374, 1992.

[76] Judith S. Olson, Gary M. Olson, Marianne Storrøsten, and Mark Carter. How a Group-editor Changes the Character of a Design Meeting as well as its Outcome. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW)*, pages 91–98. ACM, New York, NY, November 1992.

[77] Judith S. Olson, Gary M. Olson, Marianne Storrøsten, and Mark Carter. Groupwork Close Up: A Comparison of the Group Design Process With and Without a Simple Group Editor. *ACM Transactions on Information Systems*, 11(4):321–348, October 1993.

[78] Open Software Foundation, Cambridge, MA. *OSF/Motif Programmer's Guide*. Release 1.2.

[79] Arno Penzias. *Ideas and Information: Managing in a High-Tech World*. W.W. Norton & Company, Inc., New York, 1989.

[80] Michael Polanyi. *The Tacit Dimension*. Doubleday Anchor Books, New York, 1967.

[81] Colin Potts. Software-Engineering Research Revisited. *IEEE Software*, pages 18–28, September 1993.

[82] Colin Potts, Kenji Takahashi, and Annie I. Anton. Inquiry-Based Requirements Analysis. *IEEE Software*, pages 21–32, March 1994.

[83] Dave Randall and Richard Bentley. Ethnography and Systems Development: Bounding the Intersection. Tutorial Notes: CSCW'92, 1992.

[84] Gail L. Rein and Clarence A. Ellis. rIBIS: A Real-Time Group Hypertext System. *International Journal of Man-Machine Studies*, 34(3):349–367, 1991.

[85] Wolfgang Reisig. *Petri Nets: An Introduction.* Springer-Verlag, 1985.

[86] Gudula Retz-Schmidt. Various Views on Spatial Prepositions. *AI Magazine*, 9(2):95–105, 1988.

[87] Ed Rimmer. Qualitative Spatial Reasoning: A Survey. *AISB Quarterly*, Summer(88):54–61, 1994.

[88] Kenneth S. Rubin and Adele Goldberg. Object Behavior Analysis. *Communications of the ACM*, 35(9):48–62, September 1992.

[89] Kenneth S. Rubin, Patrick McClaughry, and David Pellegrini. Modeling Rules Using Object Behavior Analysis and Design. *Object Magazine*, 1994.

[90] Donald A. Schön. *The Reflective Practitioner – How Professionals Think in Action.* Basic Books, New York, 1983.

[91] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling.* John Wiley & Sons, Inc., New York, 1994.

[92] Sally Shlaer and Stephan J Mellor. *Object Lifecycles: Modeling the World in States.* Prentice Hall, Englewood Cliffs, N.J., 1992.

[93] Herbert A. Simon. The Structure of Ill Structured Problems. *Artificial Intelligence*, 4:181–201, 1973.

[94] David Sims. Review Finds Requirements Changes Plague AAS Project. *IEEE Software*, 11(2):93, March 1994.

[95] David Canfield Smith, Allen Cypher, and Jim Spohrer. KIDSIM: Programming Agents Without a Programming Language. *Communications of the ACM*, 37(7):55–67, July 1994.

[96] David E. Smyth. Widget creation library. Available via anonymous FTP at ftp.x.org in contrib/devel_tools/Wcl-2.X.tar.Z.

[97] John T. Stasko. *TANGO: A Framework and System for Algorithm Animation.* PhD thesis, Brown University, 1990.

[98] John T. Stasko. Using Direct Manipulation To Build Algorithm Animations By Demonstration. In *Proceedings of the Conference on Human Factors in Computing Systems(CHI)*, 1991.

[99] John T. Stasko and Eileen Kraemer. A Methodology for Building Application-Specific Visualizations of Parallel Programs. *Journal of Parallel and Distributed Computing*, 18(2):258–264, June 1993.

[100] B. Stroustroup. *The C++ Programming Language.* Addison-Wesley Publishing Company, 1991.

[101] M. Tambe and P. Rosenbloom. Event Tracking in Complex Multi-agent Environments. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 473–484, Orlando, FL, May 1994. Institute for Simulation and Training, University of Central Florida.

[102] John C. Tang. Findings from Observational Studies of Collaborative Work. *International Journal of Man-Machine Studies*, 34(2):143–160, 1991.

[103] United States Geological Survey. Spatial Data Transfer Standard (SDTS). file://sdts.er.usgs.gov/pub/sdts/www/html/sdtshome.html, 1992.

[104] Bill Verplank. personal communication, May 1994.

[105] Richard S. Wurman. *Follow the Yellow Brick Road: Learning to Give, Take, and Use Instructions.* Bantam Books, New York, NY, 1992.

[106] Karen Zand. *Conjure: The Alternative Coloring Book*, volume 1. Malocclusion Publishing, Box 341, 10153$\frac{1}{2}$ Riverside Drive, Toluca Lake, CA 91602, 1991.